TLA Programmatic Interface (TPI)

A Portable Document Format (PDF) version of the Tektronix TLA Programmatic Interface (TPI) online help

For TLA Version 4.2 Software

Tektronix, Inc. P.O. Box 500, Beaverton, OR 97077

TLA Programmatic Interface (TPI)
OLH152, Version 07.00

Table of Contents

What's New?

What's new in TLA TPI Version 4.2:

- Support added for the TLA7Axx Series Logic Analyzer modules
- Minor corrections to earlier versions

For information about earlier versions of the TLA TPI refer to *TLA TPI Version History*.

Introduction

The TLA programmatic interface (TPI) for the Tektronix Logic Analyzer (TLA) family is based on Microsoft's Component Object Model (COM). With TPI you can control the logic analyzer from a separate user program running on the logic analyzer or on a remote PC. Both the TLA600- and TLA700-series logic analyzers support TPI.

- In the context of TPI, the TLA application is called the server, and the program (written by the user) that controls the server through the programmable interface is called the client. The user program can be written in any language or programming environment that supports the Microsoft Component Object Model (COM), such as Microsoft Visual C++ or Microsoft Visual Basic.
- TLAScript is a standalone scripting interface to the TLA application. It is an interpreter that processes commands that are read from a script file or are manually entered by users through the TLAScript graphical interface. The TLAScript commands parallel the TPI interface. Most of the TPI methods have corresponding TLAScript commands. TLAScript has its own online help and links to the TPI online help commands.

General characteristics

The following is a list of the general characteristics of the TPI:

- TPI is consistent with programmatic interfaces exported by other Windows applications.
- All of the interfaces exported by the server are dual interfaces, supporting static and dynamic binding.
- The TLA application must be fully initialized before a client attempts to connect to it. This includes dismissing any diagnostic errors that occur at startup time. If a client attempts to connect before the application is fully initialized, it will receive an error indicating access is denied.
- A client running locally on the logic analyzer will connect to an existing instance of the server, if there is one. If the server is not running, it will be launched automatically.
- Clients may hide the server window using the programmatic interface. If the window is shown, you can directly interact with the TLA server. There will be an indicator in the status bar of the main window to show that a client is connected.
- The TLA server will continue to run after a client has disconnected. The server window is always made visible when all clients have disconnected.
- TPI operates within the main thread of the application.
- TPI supports multiple simultaneous clients. Care must be taken when using multiple clients to ensure that the clients do not interfere with one another.
- There is no provision for a TPI client to lock-out other clients.

Setting up TPI

In the following procedures, <**install directory**> refers to the directory where the TPI client is installed on your client machine. This directory is **C:\Program Files\TLA700** by default. This directory is used by both the TLA600- and TLA700-series logic analyzers.

The type library used with TPI is tla700.tlb. After you complete the following setup procedure, this file will be located in **C:\Program Files\TLA700\System\TPI** on your logic analyzer and/or in the **<install directory>\System\TPI** on your client machine.

- You have two methods of using TPI with your client application:
- You can run the client application locally on the logic analyzer. No special setup is required once the TLA application is installed. To verify that a client application can connect to the TLA server, see Running a client application on the TLA instrument.
- You can run your client application remotely across the network. In this case, both the TLA instrument and the remote host require special setup procedures. See Setting up TPI on the TLA instrument to begin this setup procedure.

Running a client application on the TLA instrument

To verify that a client application can connect to the TLA instrument, do the following steps:

1 Start the TLA application on the TLA instrument.

2 Navigate to and run the following program:
C:\Program Files\TLA 700\Samples\TPI Samples\Vc++\test client\testclient.exe
The Tektronix Test TPI Client dialog appears.

3 Click the Connect to the TLA700! button, and check that the text in the dialog changes to Connected! This indicates the test client application connected to the TLA

4 Click Exit.

Setting up TPI on the TLA instrument

If you want to run a client application remotely across the network, you must set up the TLA instrument using the following procedure:

1 Install and configure TCP/IP.

Note: If you have difficulty configuring the network setup, contact your system administrator.

2 You must choose to have either share-level access or user-level access to the TLA instrument as provided by Microsoft Windows.

Note: For TPI to work with share-level access, authentication is turned off and any COM client can call into any COM server running on the TLA instrument.

Choose one of the following:

- Share-level access
- User-level access

Share-level access

To set up the TLA instrument so that it may be shared among different users on a network, do the following steps:

1. Run **dcomcnfg.exe**.
2. Click the Default Properties tab, and set the Default Authentication Level to None.
3. Click the Applications page, and select the Tektronix TLA Application server.
4. Click the Properties button (or double-click the selected application).
5. Click the General tab, and set the Authentication Level to None.
6. Click the Location tab, and select Run application on this computer.
7. Click the Security tab, and select Use custom access permissions**.**
8. Click the Edit button.
   a) Click the Registry Value Permissions tab, and click the Add button.
   b) Scroll down the list; select and add Everyone to the list.
9. Click the Security page, and select Use custom launch permissions**.**
10. Click the Edit button.
    a) Click the Registry Value Permissions tab, and then click the Add button.
    b) Scroll down the list; select and add Everyone to the list.
11. Click the Identity page, and select The Interactive user.
12. Shut down the TLA application, and restart it before attempting to make any connections.

This completes the setup of TPI on the TLA instrument for operating with a remote client machine using share-level access.

Note: You can switch between user-level and share-level access later by redoing the procedure from step 2 of Setting up TPI on the TLA instrument.

Next, you need to set up the client machine so that it can connect to the TLA instrument. Select the appropriate setup:

- Client machine running Microsoft Windows 2000/NT
- Client machine running Microsoft Windows 95
- Client machine running Microsoft Windows 98
- Client machine running on any other platform

User-level access

The default network settings for the TLA under Windows is compatible with clients operating with user-level access. With these settings the client and server must be logged in to the same account and domain to make a connection. If this is too restrictive, we recommend that you use sharel-level access or talk to your network administrator.

To set up the TLA instrument for user-level access (default settings), do the following steps:

1 Run **dcomcnfg.exe**.
2 Click the Default Properties tab, and set the Default Authentication Level to Connect.
3 Click the Applications tab, and select the Tektronix TLA Application server. Click the Properties button (or double-click the selected application).
4 Click the General tab, and set the Authentication Level to Default.
5 Click the Location tab, and select the Run application on this computer.
6 Click the Security tab, and select the following:

- Use default access permissions
- Use default launch permissions
- Use default configuration permissions

7 Click the Identity tab, and select The launching user.
8 Shut down the TLA application, and restart it before attempting to make any connections.

This completes the set up of the TLA instrument for operation with a remote client machine using user level access or Windows 2000/NT user authentication.

Note: You can switch between user-level access and share-level access later by redoing the procedure from step 2 of Setting up TPI on the TLA instrument.

Next, you need to set up the client machine so that it can connect to the TLA instrument. Select the appropriate setup:

- Client machine running Microsoft Windows 2000/NT
- Client machine running Microsoft Windows 95
- Client machine running Microsoft Windows 98
- Client machine running on any other platform

Setting up a client machine that is running Microsoft Windows 2000/NT

After you set up the TLA instrument, you must set up the client machine using the following procedure:

Note: You may need administrative privilege to perform this procedure.

1 Install and configure TCP/IP. If you have difficulty configuring the network setup, contact your system or network administrator.
2 Load the Tektronix TLA application software CD.
3 Double-click on **TPI Client SW\Disk1\Setup.exe**.
4 Select the appropriate access type to continue with the setup procedure (you must set up the client machine to match the access level you chose for the TLA instrument):

- Share-level access
- User-level access

Setting up share-level access for Microsoft Windows 2000/NT

To set up a client machine to access a TLA instrument that is set up for share-level access, perform the following procedure (you must have administrative permissions on your computer):

1 Run **dcomcnfg.exe**.
2 Click the Default Properties tab, and set Default Authentication Level to None.
3 Click the Applications tab, and select the Tektronix TLA Application server.
4 Click the Properties button (or double-click the selected application).
5 Click the General tab, and set Authentication Level to None.
6 Click the Location tab, and select Run application on the following computer. Enter the name of the TLA instrument in the edit field.
7 Click the Security tab (if applicable), and select the Use custom access permissions; click the Edit button.
   a) Click the Registry Value Permissions tab, and then click the Add button.
   b) Scroll down the list; select and add Everyone to the list.
8 Click the Security tab (if applicable), and select Use custom launch permissions.
9 Click the Edit button.
   a) Click the Registry Value Permissions tab, and then click the Add button.
   b) Scroll down the list; select and add Everyone to the list.
10 Click the Identity tab (if present), and select The Interactive user.
11 To Verify that the setup is complete:
   a) Run **<install directory>\Samples\TPI Samples\Vc++\test\test client\testclient.exe** on the client machine.
   b) Click the button to see if the client can connect to the TLA instrument. (It may take a few minutes for the first time that you connect.)

This completes the set up of the TLA instrument and the client machine for running your client application across the network.

Setting up user-level access for Microsoft Windows 2000/NT

The default network settings for the TLA under Windows is compatible with clients operating with user-level access. With these settings the client and server must be logged in to the same account and domain to make a connection. If this is too restrictive, we recommend that you use share-level access or talk to your network administrator.

To set up a client machine to access a TLA instrument that is set up for user-level access, perform the following procedure:

1 Run **dcomcnfg.exe**.
2 Click the Default Properties tab, and set the Default Authentication Level to Connect.
3 Click the Applications tab, and select the Tektronix TLA Application server.
4 Click the Properties button (or double-click the selected application).
5 Click the General tab, and set Authentication Level to Default.
6 Click the Location tab, and select Run application on the following computer.
7 Enter the name of the TLA instrument in the edit field.
8 Click the Security tab (if present), and select the following:

- Use default access permissions
- Use default launch permissions
- Use default configuration permissions

9 Click the Identity tab (if present), and select The launching user.
10 To verify that the setup is complete:
   a) Run **<install directory>\Samples\TPI Samples\Vc++\test\test client\testclient.exe** on the client machine.
   b) Click the button to see if the client can connect to the TLA instrument. (It may take a few minutes for the first time that you connect.)

This completes the set up of the TLA instrument and the client machine for running your client application across the network.

Setting up a client machine that is running Microsoft Windows 95

After you set up the TLA instrument, you must set up the client machine using the following procedure:

1 Install and configure TCP/IP.

2 Load the Tektronix TLA application software CD.

3 Double-click TPI Client SW\Disk1\Setup.exe.

4 Download and install the following from the Microsoft web site, restarting after each installation. The Microsoft web site is at **www.microsoft.com**.

- Distributed COM (DCOM) for Microsoft Windows 95 (DCOM 95, version 1.1)
- dcomcnfg (DCOM configuration utility)

Note: The dcomcnfg utility will run only if user-level access is enabled. See step 5.

5 Select the appropriate access type to continue with the setup procedure (you must set up the client machine to match the access level you chose for the TLA instrument):

- Share-level access
- User-level access

Note: You can switch between user-level and share-level access later by redoing the procedure from Step 2 onwards.

Setting up share-level access for Microsoft Windows 95

To set up share-level access for a client machine running Microsoft Windows 95, do the following procedure:

1 In Microsoft Windows 95, click Start, select Settings, and then click Control Panel.

2 Click Network from the Control Panel, and then click the Access Control tab.

3 Choose Share-level access control, and then click OK. (If prompted, insert the Windows 95 disk or provide a file path to the stored Windows 95 files.)

4 The System Settings Change dialog box appears. Click the Yes button.

5 On the client machine, click Start and then Shut Down. In the Shut Down Windows dialog box, click Restart the Computer?, and then click the Yes button.

6 Navigate to and run **<install directory>\System\TPI\Share Level Access Client.reg**.

7. A dialog box appears, indicating successful registration. Click OK.

8 Restart the client machine using the procedure in step 5.

9 In Microsoft Windows, click Start, and then select Run. Either locate the regedit file using Browse, or enter regedit. Then click OK to run regedit.

10 Click the following registry key:
HKEY_CLASSES_ROOT\AppID\{C67DAA22-4972-11d1-9CAC-00805F0D8271}

11 Using Edit>New>StringValue, add a named value, RemoteServerName.

12 Click the new value, RemoteServerName, and select Edit>Modify. The Edit String dialog box appears.

13 Enter the computer name of the TLA instrument. This is the name used to identify the TLA on the network. Click OK.

14 To verify that the setup is complete:

a On the client machine, navigate to and run <**install directory>\Samples\TPI Samples\Vc++\test client\testclient.exe**>.
The Tektronix Test TPI Client dialog box appears.

b Click the Connect button to see if the client can connect to the TLA instrument.

Attempting Connection to TLA appears in the dialog box. (The first time you connect it may take a few minutes.) When the connection is made, the text changes to Connected!

This completes the setup of the TLA instrument and the client machine for running a client application across the network.

Note: You can switch between user-level and share-level access later by uninstalling the Tektronix TPI Client and DCOM 95 using the Windows control panel and redoing the procedure from step 2 of Setting up a client machine that is running Microsoft Windows 95.

Setting up user-level access for Microsoft Windows 95

To set up user-level access for a client machine running Microsoft Windows 95, do the following procedure:

1 In Microsoft Windows 95, select Start, Settings, and then Control Panel.

2 From the Control Panel, select Network, and then select the Access Control tab.

3 Choose User-level access control and enter the name of the domain that will be used to validate user access. Then select OK.

4 On the client machine, select Start and then Shut Down. In the Shut Down Windows dialog box, select Restart the Computer? and then press the Yes button.

5 Naviagate to and run **<install directory>\System\TPI\User Level Access Client.reg**.

6 Restart the client machine using the procedure in step 4.

7 In Microsoft Windows 95, select Start, and then select Run. Either locate the dcomcnfg file using Browse, or enter dcomcnfg. Then click OK to run dcomcnfg.

8 Double-click "Tektronix TLA Application" in the Applications page.

9 In the Location page, uncheck the "Run application on this computer" box and check the "Run application on the following computer" box. Enter the name of the TLA instrument in the edit field.

10 To verify that the setup is complete:

a Run **<install directory>\Samples\TPI Samples\Vc++\test client\testclient.exe** on the client machine.

b Click the Connect button to see if the client can connect to the TLA instrument. (The first time you connect it may take a few minutes.)

This completes the setup of the TLA instrument and the client machine for running a client application across the network.

Note: You can switch between user-level and share-level access later by uninstalling the TPI Client and DCOM 95 using the Windows control panel and redoing the procedure from step 2 of Setting up a client machine that is running Microsoft Windows 95.

Client machine running Microsoft Windows 98

After you set up the TLA instrument, you must set up the client machine using the following procedure:

1 Install and configure TCP/IP.

2 Load the Tektronix TLA logic analyzer application software CD.

3 Double-click TPI Client SW\Disk1\Setup.exe.

4 You must set up the client machine to match the access level that you chose for the TLA instrument (share-level access or user-level access).

Select the appropriate access type to continue with the setup procedure:

- Share Level Access on Windows 98
- User Level Access on Windows 98

Note: You can switch between user-level and share-level access later by redoing the procedure from Step 4 onwards.

Share Level Access on Windows 98

To set up share-level access for a client machine running Microsoft Windows 98, do the following procedure:

1 In Microsoft Windows 98, click Start, select Settings, and then click Control Panel.

2 From the Control Panel, click Network, and then click the Access Control tab.

3 Choose Share-level access control, and then click OK. (If prompted, insert the Windows 98 CD or provide a file path to the stored Windows 98 files.)

4 The System Settings Change dialog box appears. Click the Yes button.

5 On the client machine, click Start and then select Shut Down. In the Shut Down Windows dialog box, click Restart the Computer?, and then click the Yes button.

6 Navigate to and run **<install directory>\System\TPI\Share Level Access Client.reg**.

7. A dialog box appears, indicating successful registration. Click OK.

8 Restart the client machine using the procedure in step 5.

9 In Microsoft Windows, click Start, and then click Run. Either locate the regedit file using Browse, or enter regedit. Click OK to run regedit.

10 Click the following registry key:

HKEY_CLASSES_ROOT\AppID\{C67DAA22-4972-11d1-9CAC-00805F0D8271}

11 Using Edit>New>StringValue, add a named value, RemoteServerName.

12 Click the new value, RemoteServerName, and select Edit>Modify. The Edit String dialog box appears.

13 Enter the computer name of the TLA instrument. This is the name used to identify the TLA on the network. Click OK.

14 To verify that the setup is complete:

a On the client machine, navigate to and run <**install directory>\Samples\TPI Samples\Vc++\test client\testclient.exe**>.

b The Tektronix Test TPI Client dialog box appears. Click the Connect button to see if the client can connect to the TLA instrument.

c Attempting Connection to TLA appears in the dialog box. (The first time you connect it may take a few minutes.) When the connection is made, the text changes to Connected!

This completes the setup of the TLA instrument and the client machine for running a client application across the network.

Note: You can switch between user-level access and share-level access later by redoing step 4 in the procedure from Client machine running Microsoft Windows 98

User Level Access on Windows 98

To set up user-level access for a client machine running Microsoft Windows 98, do the following procedure:

1 In Microsoft Windows 98, click Start, select Settings, and then click Control Panel.

2 From the Control Panel, click Network, and then click the Access Control tab.

3 Choose User-level access control and enter the name of the domain that will be used to validate user access. Then click OK.

4 On the client machine, click Start, and then click Shut Down. In the Shut Down Windows dialog box, click Restart the Computer?, and then click the Yes button.

5 Navigate to and run **<install directory>\System\TPI\User Level Access Client.reg**.

6 Restart the client machine using the procedure in step 4.

7 In Microsoft Windows 98, click Start, and then click Run. Either locate the dcomcnfg file using Browse, or enter dcomcnfg. Click OK to run dcomcnfg.

8 Double-click "Tektronix TLA application" in the Applications page.

9 In the Location page, clear the "Run application on this computer" box and check the "Run application on the following computer" box. Enter the name of the TLA instrument in the edit field.

10 To verify that the setup is complete:

a Run **<install directory>\Samples\TPI Samples\Vc++\test client\testclient.exe** on the client machine.

b Click the Connect button to see if the client can connect to the TLA instrument. (The first time you connect it may take a few minutes.)

This completes the setup of the TLA instrument and the client machine for running a client application across the network.

Note: You can switch between user-level and share-level access later by redoing the procedure from step 2 of Client machine running Microsoft Windows 98.

Client instrument running on any other platform

If the client application requires use of the type library, you can generate it on your platform using tla700.odl in **C:\Program Files\TLA 700\System\TPI\src**.

Perform the following steps:

1 Ensure that DCOM is working on your platform.

2 Merge **C:\Program Files\TLA 700\System\TPI\Client.reg** into your registry.

3 M erge **C:\Program Files\TLA 700\System\TPI\Share Level Access Client.reg** or **User Level Access Client.reg** into your registry, depending on the type of access control you chose for the TLA instrument.

4 Add a string value named RemoteServerName to the key HKEY_CLASSES_ROOT\AppID\{275FF661-6554-11d3-9D63-00805F0D8271}

5 Enter the computer name of the logic analyzer as this string value. This is the name used to identify the logic analyzer on the network.

This completes the setup of the logic analyzer for operating with a remote client machine using a platform other than Microsoft Windows.

Connecting to the TLA server

Client applications connect to the TLA server by creating an Application Object. For example, in Microsoft Visual Basic:

```
'Establish connection to TLA.
Dim App As Object
Set App = CreateObject("TLA700.Application")
```

Once the Application Object is created, the client can call methods on it to get references to System and Module Objects.

Disconnecting from the TLA server

A client application connected to the TLA server can disconnect by deleting the reference to the Application Object. For example, in Microsoft Visual Basic:

```
'Disconnect from TLA.
Set App = Nothing
```

References to any System or Module Objects that were obtained must also be deleted.

Programming examples

All interfaces exported by the TLA server are dual interfaces that have the characteristics of both dispatch and vtable interfaces.

The dispatch side of a dual interface uses run time (dynamic) binding to resolve method calls. This is mostly used in interpretive and scripting environments where static binding using header files and type libraries cannot be performed.

The vtable side of a dual interface uses static binding to resolve method calls. This is accomplished using header files and type libraries.

See the following examples:

- Example of a Microsoft Visual Basic client using dispatch interfaces
- Example of a Microsoft Visual Basic client using vtable interfaces

Additional code examples for specific methods use the dispatch portion of each dual interface.

Example of a Microsoft Visual Basic client using dispatch interfaces

This client example uses the dynamic (dispatch) part of the dual interfaces.

```
Dim App As Object
Dim System As Object
Dim LA As Object
Dim Status As Long
Dim BytesPerSample As Long
Dim S As String
Dim Data As Variant

Private Sub Form_Load()

    ‘Connect to server.
    Set App = CreateObject("TLA700.Application")
‘Get system pointer.
    Set System = App.GetSystem
‘NOTE: To load a system, fill in system path.
    System.LoadSystem "<path>"
‘NOTE: Fill in the slot no. of your LA here.
    Set LA = System.GetModuleBySlot(<slot>)
‘Run acquisition.
    Run
‘Get data.
    GetData
‘Show data.
    ShowData
End Sub

Private Sub Run()
    ‘Do acquisition. Wait for it to complete.
    System.Run
    Do
        Status = System.GetRunStatus
    Loop While(Status = 0)
End Sub

Private Sub GetData()
    ‘Get the first ten samples (grouped) in the main data set
    ‘in ASCII.
    ‘NOTE: Change group list if required.
    BytesPerSample = LA.DefineDataFormat(0,
"GroupList:A3,A2,Timestamp", 1)
    Data = LA.GetData(0, 10)
End Sub

Private Sub ShowData()
    ‘Show data.
    For I = 0 To 9
        S = Data(I)
        DataList.AddItem(S)
    Next I
End Sub
```

Example of a Microsoft Visual Basic client using vtable interfaces

This client example uses the vtable (static) part of the dual interfaces.

```
Dim App As IApplication
Dim System As ISystem
Dim LA As ILAModule
Dim Status As Long
Dim BytesPerSample As Long
Dim S As String
Dim Data As Variant

Private Sub Form_Load()
    'Connect to server.
    Set App = CreateObject("TLA700.Application")
    'Get system pointer.
    Set System = App.GetSystem
'NOTE: To load a system, fill in system path.
    System.LoadSystem("<path>")
'NOTE: Fill in the slot no. of your LA here.
    Set LA = System.GetModuleBySlot(<slot>)
    'Run acquisition.
    Run
    'Get data.
    GetData
    'Show data.
    ShowData
End Sub

Private Sub Run()
    'Do acquisition. Wait for it to complete.
    System.Run
    Do
        Status = System.GetRunStatus
    Loop While(Status = 0)
End Sub

Private Sub GetData()

    'Get the first ten samples (grouped) in the main data set
    'in ASCII.
    'NOTE: Change group list if required.
    BytesPerSample = LA.DefineDataFormat(0,"GroupList:A3,A2,Timestamp",
1)
    Data = LA.GetData(0, 10)
End Sub

Private Sub ShowData()
    'Show data.
    For I = 0 To 9
        S = Data(I)
        DataList.AddItem(S)
    Next I
End Sub
```

Error handling

All methods in all interfaces of TPI return an HRESULT (or SCODE). Refer to the file "tla700error.h" (the path is **<install directory>\System\TPI\Src\tla700error.h>**) for possible error codes.

Additional error information is communicated as follows:

- Objects that use the dispatch portion of the dual interface can use the exception information argument of the IDispatch::Invoke method.
- Objects that use the vtable portion of the dual interface can use error objects. When an HRESULT indicates an error, the client can call the standard function GetErrorInfo() to get more detailed information about the error.
- When a method returns an error, output arguments are undefined and should not be used.

Server Message Boxes

In the TLA graphical user interface, there are instances where you are asked to confirm a particular operation. For example, before loading a saved system, you are asked whether the current system should be saved before the load operation.

Since it is not possible to ask questions through TPI, the application always proceeds with the original operation as though the question were never asked. In the previous example, the load operation proceeds without saving the current system.

Modal message boxes that are normally displayed in the TLA user interface will not be displayed when a client is connected to the server.

Slot numbers for expansion mainframes

In TPI, the slot numbers for expansion mainframes are specified by extending the slot numbers for the mainframe. For example, consider a system configuration consisting of a TLA720 benchtop mainframe with two expansion frames each containing 13 slots:

- Mainframe: Slots 0 - 12
- Expansion 1 Slots 13 - 25
- Expansion 2 Slots 26 – 38

External Oscilloscope modules do not have slot numbers.

Tips for improving LA module data transfer performance

The data transfer rates (in KB/s) that you achieve using ILAModule::GetData() depend on various factors. The following guidelines will improve data transfer performance:

- **Buffer size.** When a client invokes ILAModule::GetData(), the TLA server allocates and returns a data buffer to the client. The size of this buffer affects the efficiency of the data transfer operation. Using too small a buffer reduces performance because of the added overhead of additional calls to transfer a fixed amount of data. Using too large a buffer can cause the system to exceed the available physical memory. When this happens, virtual memory must be made available by swapping physical memory pages to the hard disk, which again reduces performance. Optimal performance is achieved using buffers that are as large as possible without exceeding the available physical memory.

  The size of the buffer is determined by the number of samples requested. For systems with 32M of , and only the TLA application running, the optimum buffer size is typically between 1,300,000 - 2,600,000 bytes. For the 136-channel TLA7N4 LA module, this is approximately within the range of 50,000 - 100,000 samples for raw data (since there are 26 bytes per sample). See Binary data formats for LA modules and Text data formats for LA modules for more information on the number of bytes per sample for other module types.

With larger systems, or when additional applications are running, it can be difficult to determine how much physical memory is available. One technique for determining this number empirically is to use to do data transfers of varying size, while monitoring memory use using the Win2000 performance monitor. The performance monitor can be found at StartMenu>Programs>Administrative Tools>Performance. The measurement called "Available Mbytes" shows the amount of physical memory remaining. The objective is to find out how much data you can transfer in one call, without causing this measurement to hit zero.

- **Amount of TLA and client machine main memory.** As expected, the larger the amount of random access memory (RAM), the better the performance. This difference is more pronounced as the number of samples requested increases. It is less pronounced when an optimum number of samples is requested (see the previous bulleted item, Buffer size).

- **Data formats.** A TPI client application has a choice of data formats. For details, see ILAModule::DefineDataFormat().

  Binary Text. In general, binary data transfer is faster than text data transfer. Binary data transfer may, however, require some effort on the part of the TPI client to process the binary data.

  Raw versus Grouped. If the TPI client requires most of the information in a given sample of data, a raw data is more efficient than a grouped format. However, the raw format requires more post-transfer processing by the TPI client and gives the user less flexibility in what data is transferred. If you require very little information out of a given sample, the grouped format may be more appropriate.

  To clarify difference between Binary versus Text and Raw versus Grouped, look at the following scenarios:

  Scenario 1 User A is interested in getting data as fast as possible and is willing to filter and process data in any format.

  The appropriate choice in this scenario would probably be the RawWithTimestamp format.

  Scenario 2 User B is interested in only one particular group. The user would like the data to be formatted. The speed of data transfer is not a critical issue.

The appropriate choice in this scenario would probably be the GroupList format, just the group of interest.

- **LA** module **characteristics.** The narrower LA modules (fewer channels) tend to have slower transfer rates than the wider modules.

- **Client application/programming environment.** The efficiency of the data transfer is affected by the way the TPI client application is written and/or the programming environment used for writing the client application. Remember that it is the responsibility of the TPI client application to free memory allocated for data buffers during data transfer.

- **Network conditions.** Data transfer using TPI using DCOM over the network gives approximately half the performance of local COM data transfer. This may vary depending upon the network traffic conditions.

Objects & Interfaces

Objects and Interfaces

The TPI consists of four types of objects:

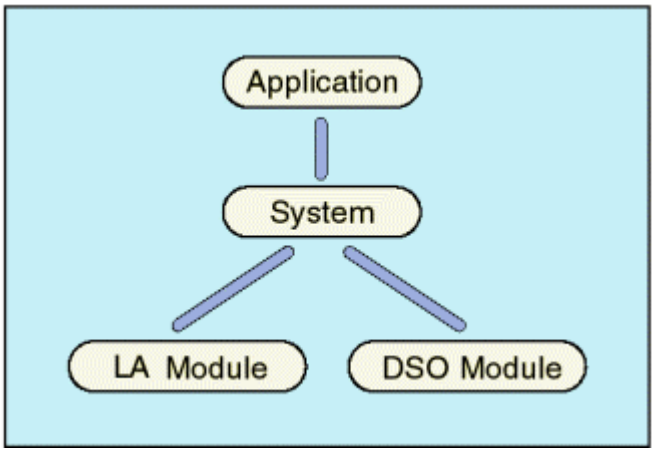
- Application
- System
- LAModule
- DSOModule

You an Application Object to initially connect to the TLA application and to subsequently obtain a reference to a system object. The application object exports a single interface called IApplication.

The system provides methods for configuration, run control, and save and load operations. Every client must obtain a reference to a System Object before it can obtain references to Module Objects. The System Object exports a single interface called ISystem.

Module objects provide methods for module configuration, obtaining acquisition statistics, and retrieving acquisition . Logic analyzer module objects export a single interface called ILAModule; DSO module objects export a single interface called IDSomodule.

Unless otherwise specified, all methods are synchronous and wait for the completion of the operation before . The syntax of the TPI methods is described in Microsoft Object Description Language (ODL). The object hierarchy is shown below:



Application Object

The Object supports the following methods:

IApplication

HRESULT GetSystem(ppDispatch)

HRESULT ShowWindow(Show)

System Object

The System supports the following methods:

ISystem

Properties:

VARIANT_BOOL Repetitive

long RunCount (read-only)

long ExternalSignalIn

long ExternalSignalOut

VARIANT_BOOL ExternalSignalOutLowTrue

Configuration Methods:

HRESULT GetNumModuleSlots(pNumSlots)

HRESULT GetFirstModuleSlot(pSlot)

HRESULT GetSWVersion(pVersion)

HRESULT GetDiagCalStatus(pDiagCalStatus)

HRESULT GetModuleTypeBySlot(Slot, pModuleType)

HRESULT GetModulePropertiesBySlot(Slot, pModuleProperties)

HRESULT GetModuleBySlot(Slot, ppDispatch)

HRESULT GetModuleByName(ModuleName, ppDispatch)

HRESULT GetModuleNames(pModuleNames)

HRESULT GetModuleSlotByName(ModuleName, pSlot)

Load and Save Methods:

HRESULT LoadSystem(SystemPath)

HRESULT SaveSystem(SystemPath, UserComment, SaveData)

HRESULT DefineRangeSymbolOptions(FileFormat, SymbolTypes, Reserved, Bound1, Bound2, OffsetType, SymbolOffset)

HRESULT LoadSymbolFile(SymbolFilePath)

Run Control and Status Methods:

HRESULT Run()

HRESULT Stop()

HRESULT GetRepetitiveStopReason (pStopReason)

HRESULT GetRunStatus(pRunStatus)

LAModule Object

The LAModule supports the following methods:

ILAModule

Properties:

BSTR Name

VARIANT_BOOL Enabled

long MemoryDepth

Load and Save Methods:
HRESULT LoadModule(ModulePath, ModuleName)
HRESULT LoadTrigger(ModulePath, ModuleName)
HRESULT SaveModule(ModulePath, UserComment, SaveData)

Setup Methods:
HRESULT DeleteChannelGroup(UserChannelGroupName)
HRESULT GetChannelGroup(UserChannelGroupName, ChannelNameList)
HRESULT GetChannelName(HWChannelName, UserChannelName)
HRESULT GetGroupNames(GroupNames)
HRESULT GetGroupSize(GroupNames, pGroupSize)
HRESULT SetEventValue(EventID, EventValue)
HRESULT SetChannelGroup(UserChannelGroupName, ChannelNameList)
HRESULT SetChannelName(HWChannel Name, UserChannelName)
HRESULT SetTriggerPosition(Position)

Acquisition and Statistics Methods:
HRESULT GetBeginTime(DataSet, pTimeHighWord, pTimeLowWord)
HRESULT GetCounterValue(CounterID, pCounterHighWord, pCounterLowWord)
HRESULT GetEndTime(DataSet, pTimeHighWord, pTimeLowWord)
HRESULT GetNumSamples(DataSet, pNumSamples)
HRESULT GetStartTime(pDate)
HRESULT GetTimerValue(TimerID, pTimerHighWord, pTimerLowWord)
HRESULT GetTriggerSample(DataSet, pTriggerSample)
HRESULT GetTriggerTime(DataSet, pTimeHighWord, pTimeLowWord)

Acquisition Data Methods:
HRESULT DefineDataFormat(DataSet, Components, DataType, pBytesPerSample)
HRESULT ExportData(FirstSample, NumSamples, DataPath, Mode)
HRESULT GetBytesPerSample(pBytesPerSample)
HRESULT GetData(FirstSample, NumSamples, pData)
HRESULT GetTimestampMultiplier(pMultiplier)

DSOModule Object

The following methods are for the IDSOModule Object: [Unless otherwise stated, all DSOModule object properties and methods apply to External oscilloscope modules as well as Internal DSO modules.]

IDSOModule

Properties:

BSTR Name

VARIANT_BOOL Enabled

Load and Save Methods:

HRESULT LoadModule(ModulePath, ModuleName)

HRESULT SaveModule(ModulePath, UserComment, SaveData)

Acquisition and Status Methods:

HRESULT GetNumSamples(pNumSamples)

HRESULT GetTriggerSample(pTriggerSample)

HRESULT GetTriggerTime(pTimeHighWord, pTimeLowWord)

HRESULT GetBeginTime(pTimeHighWord, pTimeLowWord)

HRESULT GetEndTime(pTimeHighWord, pTimeLowWord)

HRESULT GetStartTime(pDate)

Acquisition Data Methods:

HRESULT GetDataRange(Channel, pRange)

HRESULT GetDataOffset(Channel, pOffset)

HRESULT GetDataSamplePeriod(pPeriodHighWord, pPeriodLowWord)

HRESULT DefineDataFormat(Components, DataType)

HRESULT GetData(FirstSample, NumSamples, pData)

HRESULT ExportData(FirstSample, NumSamples, DataPath, Mode)

Methods by Alphabetical Listing

Methods by Alphabetical Listing

IApplication::GetSystem

Description

This method returns the interface pointer for the System Object.

ODL Syntax

```
HRESULT GetSystem( [out, retval] IDispatch** ppDispatch )
```

Arguments

`ppDispatch` The interface pointer for the System Object.

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
TLA700_E_OUT_OF_MEMORY	There is not enough memory to perform the operation.
TLA700_E_FAILED	The operation was unsuccessful.

Example

```
(Microsoft Visual Basic)

Dim App As Object
Dim Sys As Object

Set App = CreateObject("TLA700.Application")
 'Get system.
Set Sys = App.GetSystem
```

IApplication::ShowWindow

Description

This method shows or hides the TLA server's application window.

ODL Syntax

```
HRESULT  ShowWindow(  [in]  long  Show  )
```

Arguments

`Show` This flag takes one of the values in the following table:

Value	Description
TLA700_HIDE_WINDOW (0)	Hide the server window.
TLA700_SHOW_WINDOW (1)	Show the server window.

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
TLA700_E_INVALID_SHOW	Invalid “Show” argument.
TLA700_E_OUT_OF_MEMORY	There is not enough memory to perform the operation.
TLA700_E_FAILED	The operation was unsuccessful.

Example

```
(Microsoft Visual Basic)

Dim App As Object

Set App = CreateObject("TLA700.Application")
 'Hide the window.
App.ShowWindow 0
```

Remarks

The application window is shown by default when a client connects to the server.

IDSOModule::DefineDataFormat

Description

This method specifies the format of the data which is returned in subsequent queries for acquisition data (using IDSOModule::GetData() and IDSOModule::ExportData()).

ODL Syntax

```
HRESULT  DefineDataFormat(  [in]  BSTR  Components,
                            [in]  long  DataType  )
```

Arguments

`Components` This takes one of the following forms:

`"AllChannels"` Data for all channels is returned.

`"ChannelList:<channel>,<channel>,<channel>…"` Specifies exactly which channels to return and the order in which they are to be returned. The same channel may be included more than once. Channels are specified using their channel numbers (for example, 1, 2, 3, and 4).

For details on the format of these components, refer to the following topics:

`DataType` This takes one of the values from the following table:

Value	Description
TLA700_BINARY (0)	Binary
TLA700_TEXT_SPACE (1)	ASCII, space delimiter
TLA700_TEXT_TAB (2)	ASCII, tab delimiter
TLA700_TEXT_COMMA (3)	ASCII, comma delimiter

For binary format, data is returned as an array of short values (16 bits). There are no delimiters between samples.

For text format, data is returned as an array of strings, one per sample. Data is returned in volts without the unit characters appended.

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
TLA700_E_INVALID_COMPONENTS	Invalid “Components” argument.
TLA700_E_INVALID_DATA_TYPE	Invalid “DataType” argument.
TLA700_E_OUT_OF_MEMORY	There is not enough memory to perform the operation.
TLA700_E_FAILED	The operation was unsuccessful.

Example

```
(Microsoft Visual Basic)

Dim App As Object
Dim Sys As Object
Dim DSO As Object
Dim Slot As Long
Set App = CreateObject("TLA700.Application")
 ‘Get system.
Set Sys = App.GetSystem
 ‘Get module.
Set DSO = Sys.GetModuleBySlot(Slot)
 ‘Get data in binary format.
DSO.DefineDataFormat "ChannelList:1",0
```

IDSOModule::Enabled

Description

This property allows the client to enable/disable the logical module.

ODL Syntax

```
[
   propget
]
HRESULT Enabled( [out, retval] VARIANT_BOOL* pEnabled )

[
   propput
]
HRESULT Enabled( [in] VARIANT_BOOL Enabled )
```

Arguments (propget)

`pEnabled` - The enabled status of the logical module.

Arguments (propput):

`Enabled` - The enabled status of the logical module.

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
TLA700_E_SYSTEM_RUNNING	The operation cannot be performed when the system is acquiring data.
TLA700_E_OUT_OF_MEMORY	There is not enough memory to perform the operation.
TLA700_E_FAILED	The operation was unsuccessful.
TLA700_E_MODAL_DIALOG_OPEN	This operation was disallowed because a modal dialog is open.

Example

```
(Microsoft Visual Basic)

Dim App As Object
Dim Sys As Object
Dim DSO As Object
Dim Enabled As Boolean

Set App = CreateObject("TLA700.Application")
 'Get system.
Set Sys = App.GetSystem
 'Get module in slot 3.
Set DSO = Sys.GetModuleBySlot(3)
 'Get enabled status.
Enabled = DSO.Enabled
 'Enable module.
DSO.Enabled = True
```

IDSOModule::ExportData

Description

This method exports the specified acquisition data to a file.

ODL Syntax

```
HRESULT  ExportData(  [in]  long  FirstSample,
                      [in]  long  NumSamples,
                      [in]  BSTR  DataPath,
                      [in]  long  Mode  )
```

Arguments

`FirstSample` The sample number of the first sample to export.

`NumSamples` The number of samples to export.

`DataPath` The complete path of the file to export to.
For example: **"C:\My Documents\My Data.txt"**
"C:\My Documents\My Data.tbf" for binary format

`Mode` This argument takes one of the following values:

Value	Description
TLA700_APPEND (0)	Causes the new data to be appended to whatever is already in the file.
TLA700_OVERWRITE (1)	Causes data existing in the file to be overwritten.

The composition and format of the data must be specified by the IDSOModule::DefineDataFormat() call.

If the data type is TLA700_BINARY, the file is a stream of 16-bit short values.

If the data type is TLA700_TEXT_*, the file consists of a stream of newline separated strings, one string per sample.

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
TLA700_E_INVALID_DATA_FORMAT	A valid data format must be defined before calling this method. The data format you defined previously may no longer be valid.
TLA700_E_VALUE_NOT_AVAILABLE	The required value(s) are not available.
TLA700_E_INVALID_FIRST_SAMPLE	Invalid “FirstSample” argument.
TLA700_E_INVALID_NUM_SAMPLES	Invalid “NumSamples” argument.
TLA700_E_INVALID_MODE	Invalid “Mode” argument.
TLA700_E_EXPORT_INVALID_FILE	An error occurred opening this file for writing.
TLA700_E_EXPORT_ERROR	An error occurred during the export operation.
TLA700_E_OUT_OF_MEMORY	There is not enough memory to perform the operation.
TLA700_E_FAILED	The operation was unsuccessful.

Example

```
(Microsoft Visual Basic)

Dim App As Object
Dim Sys As Object
Dim DSO As Object
Dim Slot As Long
Dim NumSamples As Long

Set App = CreateObject("TLA700.Application")
 'Get system.
Set Sys = App.GetSystem
 'Get module.
Set DSO = Sys.GetModuleBySlot(Slot)
 'Export data in text format.
DSO.DefineDataFormat "ChannelList:1,2",1
DSO.ExportData 0,NumSamples,"C:\export.txt", 0
```

Remarks

The value specified for FirstSample must be greater than or equal to zero and less than the total number of samples acquired. NumSamples must be greater than zero, and FirstSample + NumSamples must be less than or equal to the total number of samples acquired. If these conditions are not met, an error code is returned.

A data format must be defined using DefineDataFormat() before calling this method.

All file paths without machine qualifiers refer to drives mapped on the TLA instrument.

If the data is of type TLA700_BINARY, the actual data value in volts corresponding to each binary (short) value returned may be calculated as follows:

<Data value in volts> = ((<vertical range in volts> / 64512) * <binary data value>) + <vertical offset in volts>

The vertical range and offset may be obtained using IDSOModule::GetDataRange() and IDSOModule::GetDataOffset().

IDSOModule::GetBeginTime

Description

This method returns the begin time of the module.

ODL Syntax

```
HRESULT  GetBeginTime(  [out]  long*  pTimeHighWord,
                        [out]  long*  pTimeLowWord  )
```

Arguments

`pTimeHighWord`	The higher word of the begin time in picoseconds.
`pTimeLowWord`	The lower word of the begin time in picoseconds.

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
TLA700_E_SYSTEM_RUNNING	The operation cannot be performed when the system is acquiring data.
TLA700_E_OUT_OF_MEMORY	There is not enough memory to perform the operation.
TLA700_E_FAILED	The operation was unsuccessful.

Example

```
(Microsoft Visual Basic)

Dim App As Object
Dim Sys As Object
Dim DSO As Object
Dim Slot As Long
Dim TimeHigh As Long
Dim TimeLow As Long
Set App = CreateObject("TLA700.Application")
 'Get system.
Set Sys = App.GetSystem
 'Get module.
Set DSO = Sys.GetModuleBySlot(Slot)
 'Get begin time.
DSO.GetBeginTime TimeHigh, TimeLow
```

Remarks

Begin time is a 64-bit value. This method returns this value in the form of two long values representing the high and low words. Though the low word is returned in a signed long value, it is to be treated as an *unsigned* value.

Begin time is equivalent to the timestamp of the first sample in the acquisition buffer. Its value is the number of pico-seconds that have elapsed since the start of the acquisition. The start of the acquisition is the moment at which the system is enabled to begin acquiring data, however, there will always be some elapsed time before actual data acquisition begins. Other factors such as acquisition buffer depth, clock rate, storage qualification or delay while waiting for a trigger condition to occur can also affect the period of time between the start of the acquisition and the time of the first sample in the storage buffer.

The Begin time value is reported in terms of pico-seconds, and has been adjusted to include the frame offset and user specified time alignment values, if any. If additional, user defined, per-channel offsets are specified, those offsets are not reflected in the Begin time value returned by this method. For a further explanation of these values, see the discussion of Timestamp values in the DataFormat section near the end of this document.

IDSOModule::GetData

Description

This method returns the specified acquisition data.

ODL Syntax

```
HRESULT  GetData(  [in]  long  FirstSample,
                   [in]  long  NumSamples,
                   [out,  retval]  VARIANT*  pData  )
```

Arguments

`FirstSample` The sample number of the first sample to return. Sample numbers start with 0.

`NumSamples` The number of samples to return.

`pData` The acquisition data. The composition and format of the data is determined by the IDSOModule::DefineDataFormat() call.

Data is returned as a VARIANT. The variant is of type VT_ARRAY and points to a SAFEARRAY. The SAFEARRAY has a dimension of one and has NumSamples entries.

If the data type is TLA700_BINARY, the SAFEARRAY is of element type VT_I2 (short).

If the data type is TLA700_TEXT_* the element type is VT_BSTR (string).

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
TLA700_E_INVALID_DATA_FORMAT	A valid data format must be defined before calling this method. The data format you defined previously may no longer be valid.
TLA700_E_ VALUE_NOT_AVAILABLE	The required value(s) are not available.
TLA700_E_INVALID_FIRST_SAMPLES	Invalid "FirstSample" argument.
TLA700_E_INVALID_NUM_SAMPLES	Invalid "NumSamples" argument.
TLA700_E_OUT_OF_MEMORY	There is not enough memory to perform the operation.
TLA700_E_BUFFER_LIMIT_EXCEEDED	The size of the requested data buffer exceeded the maximum limit.
TLA700_E_FAILED	The operation was unsuccessful.

Example

```
(Microsoft Visual Basic)

Dim App As Object
Dim Sys As Object
Dim DSO As Object
Dim Slot As Long
Dim NumSamples As Long
Dim DataArray As Variant

Set App = CreateObject("TLA700.Application")
 'Get system.
Set Sys = App.GetSystem
 'Get module.
Set DSO = Sys.GetModuleBySlot(Slot)
 'Get main data in binary format.
DSO.DefineDataFormat "ChannelList:1,2",0
DataArray = DSO.GetData(0, NumSamples)
 ' Process samples.
For I = 0 To NumSamples - 1
      `Process DataArray(I)
Next I
```

Remarks

The value specified for FirstSample must be greater than or equal to zero and less than the total number of samples acquired. NumSamples must be greater than zero, and FirstSample + NumSamples must be less than or equal to the total number of samples acquired. If these conditions are not met, an error code is returned.

A data format must be defined using IDSOModule::DefineDataFormat() before calling this method.

The TLA700 server allocates the space for the array of data. The client is responsible for freeing the space when it is no longer in use.

If the data is of type TLA700_BINARY, the actual data value in volts corresponding to each binary (short) value returned may be calculated as follows:

<Data value in volts> = ((<vertical range in volts> / 64512) * <binary data value>) + <vertical offset in volts>

The vertical range and offset may be obtained using IDSOModule::GetDataRange and IDSOModule::GetDataOffset.

IDSOModule::GetDataOffset

Description

This method returns the vertical offset for the current acquisition data of a channel. This value is used to interpret binary acquisition data obtained from the module.

ODL Syntax

```
HRESULT  GetDataOffset([in]  long  Channel  ,
                       [out, retval]  double*  pOffset  )
```

Arguments

`Channel` The number of the channel. This value should be in the range 1…n where n is the total number of channels in the DSO.

`pOffset` The vertical offset of the channel in volts.

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
TLA700_E_INVALID_CHANNEL	Invalid “Channel” argument.
TLA700_E_VALUE_NOT_AVAILABLE	The required value(s) are not available.
TLA700_E_OUT_OF_MEMORY	There is not enough memory to perform the operation.
TLA700_E_FAILED	The operation was unsuccessful.

Example

```
(Microsoft Visual Basic)

Dim App As Object

Dim Sys As Object
Dim DSO As Object
Dim Slot As Long
Dim Offset As Double

Set App = CreateObject("TLA700.Application")
 ‘Get system.
```

```
Set Sys = App.GetSystem
 'Get module.
Set DSO = Sys.GetModuleBySlot(Slot)
 'Get data offset.
Offset = DSO.GetDataOffset(1)
```

Remarks

The value returned is the vertical offset for the data currently in the module.

IDSOModule::GetDataRange

Description

This method returns the vertical range for the current acquisition data of a channel, which is used to interpret binary acquisition data obtained from the module.

ODL Syntax

```
HRESULT  GetDataRange(  [in]  long  Channel,
                        [out, retval]  double*  pRange  )
```

Arguments

`Channel` The number of the channel. This value should be in the range 1…n, where n is the total number of channels in the DSO.

`pRange` The vertical range of the channel in volts.

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
TLA700_E_INVALID_CHANNEL	Invalid "Channel" argument.
TLA700_E_VALUE_NOT_AVAILABLE	The required value(s) are not available.
TLA700_E_OUT_OF_MEMORY	There is not enough memory to perform the operation.
TLA700_E_FAILED	The operation was unsuccessful.

Example

```
 (Microsoft Visual Basic)

 Dim App As Object
 Dim Sys As Object
 Dim DSO As Object
 Dim Slot As Long
 Dim Range As Double

 Set App = CreateObject("TLA700.Application")
  'Get system.
 Set Sys = App.GetSystem
  'Get module.
 Set DSO = Sys.GetModuleBySlot(Slot)
  'Get data range.
Range = DSO.GetDataRange(1)
```

Remarks

The value returned is the vertical range for the data currently in the module.

IDSOModule::GetDataSamplePeriod

Description

This method returns the sample period for the module's current acquisition data.

ODL Syntax

```
HRESULT  GetSamplePeriod(  [out]  long*  pPeriodHighWord,
                           [out]  long*  pPeriodLowWord  )
```

Arguments

`pPeriodHighWord` The higher word of the period in picoseconds.

`pPeriodLowWord` The lower word of the period in picoseconds.

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
TLA700_E_VALUE_NOT_AVAILABLE	The required value(s) are not available.
TLA700_E_OUT_OF_MEMORY	There is not enough memory to perform the operation.
TLA700_E_FAILED	The operation was unsuccessful.

Example

```
(Microsoft Visual Basic)

Dim App As Object
Dim Sys As Object
Dim DSO As Object
Dim Slot As Long
Dim PeriodHigh As Long
Dim PeriodLow As Long

Set App = CreateObject("TLA700.Application")
 'Get system.
Set Sys = App.GetSystem
 'Get module.
Set DSO = Sys.GetModuleBySlot(Slot)
 'Get sample period.
DSO.GetDataSamplePeriod PeriodHigh, PeriodLow
```

Remarks

The value returned is the sample period for the data currently in the module. It is a 64-bit value. This method returns the value in the form of two long values representing the high and low words. Although the low word is returned as a signed long value, treat it as an unsigned value.

IDSOModule::GetEndTime

Description

This method returns the end time of the module.

ODL Syntax

```
HRESULT  GetEndTime(  [out]  long*  pTimeHighWord,
                      [out]  long*  pTimeLowWord  )
```

Arguments

`pTimeHighWord` The higher word of the end time in picoseconds.

`pTimeLowWord` The lower word of the end time in picoseconds.

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
TLA700_E_SYSTEM_RUNNING	The operation cannot be performed when the system is acquiring data.
TLA700_E_OUT_OF_MEMORY	There is not enough memory to perform the operation.
TLA700_E_FAILED	The operation was unsuccessful.

Example

```
(Microsoft Visual Basic)

Dim App As Object
Dim Sys As Object
Dim DSO As Object
Dim Slot As Long
Dim TimeHigh As Long
Dim TimeLow As Long

Set App = CreateObject("TLA700.Application")
 'Get system.
Set Sys = App.GetSystem
 'Get module.
Set DSO = Sys.GetModuleBySlot(Slot)
 'Get end time.
DSO.GetEndTime TimeHigh, TimeLow
```

Remarks

End time is a 64-bit value. This method returns this value in the form of two long values representing the high and low words. Though the low word is returned in a signed long value, it is to be treated as an *unsigned* value.

End time is equivalent to the timestamp of the last sample in the acquisition buffer. Its value is the number of pico-seconds that have elapsed since the start of the acquisition. The start of the acquisition is the moment at which the system is enabled to begin acquiring data, however, there will always be some elapsed time before actual acquisition begins. Other factors such as acquisition buffer depth, clock rate, storage qualification or delay while waiting for a trigger condition to occur can also lengthen the period of time between the start of the acquisition and the time of the last sample in the storage buffer.

The End time value is reported in terms of pico-seconds, and has been adjusted to include the frame offset and user specified time alignment values, if any. If additional, user defined, per-channel offsets are specified, those offsets are not reflected in the End time value returned by this method. For a further explanation of these values, see the discussion of Timestamp values in the DataFormat section near the end of this document.

IDSOModule::GetNumSamples

Description

This method returns the number of samples in the acquisition memory of the module.

ODL Syntax

```
HRESULT GetNumSamples( [out, retval] long* pNumSamples )
```

Arguments

`pNumSamples` The number of samples in the acquisition memory of this module.

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
TLA700_E_SYSTEM_RUNNING	The operation cannot be performed when the system is acquiring data.
TLA700_E_OUT_OF_MEMORY	There is not enough memory to perform the operation.
TLA700_E_FAILED	The operation was unsuccessful.

Example

```
(Microsoft Visual Basic)

Dim App As Object
Dim Sys As Object
Dim DSO As Object
Dim Slot As Long
Dim NumSamples As Long

Set App = CreateObject("TLA700.Application")
 'Get system.
Set Sys = App.GetSystem
 'Get module.
Set DSO = Sys.GetModuleBySlot(Slot)
 'Get number of samples.
NumSamples = DSO.GetNumSamples
```

Remarks

This method will return a value of zero if no data is available.

IDSOModule::GetStartTime

Description

This method returns the start date and start time for the module acquisition.

ODL Syntax

```
HRESULT GetStartTime( [out, retval] VARIANT* pDate )
```

Arguments

`pDate` The start date and start time of the module acquisition. This is returned as a VARIANT. The variant is of type VT_DATE.

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
TLA700_E_SYSTEM_RUNNING	The operation cannot be performed when the system is acquiring data.
TLA700_E_VALUE_NOT_AVAILABLE	The required value(s) are not available.
TLA700_E_OUT_OF_MEMORY	There is not enough memory to perform the operation.
TLA700_E_FAILED	The operation was unsuccessful.

Example

```
(Microsoft Visual Basic)

Dim App As Object
Dim Sys As Object
Dim DSO As Object
Dim StartTime As Date

Set App = CreateObject("TLA700.Application")
 'Get system.
Set Sys = App.GetSystem
'Get module in slot 3.
Set DSO = Sys.GetModuleBySlot(3)
 'Get start time.
StartTime = DSO.GetStartTime
```

Remarks

The start of an acquisition is not the same as the first sample of data. The start of an acquisition is the moment at which the system is enabled to begin acquiring data. It takes some finite amount of time after the start of the acquisition before a module can actually store a sample of data.

IDSOModule::GetTriggerTime

Description

This method returns the time stamp for the trigger sample of the acquired data for the module.

ODL Syntax

```
HRESULT  GetTriggerTime(  [out]  long*  pTimeHighWord,
                          [out]  long*  pTimeLowWord  )
```

Arguments

`pTimeHighWord` The higher word of the trigger time in picoseconds.

`pTimeLowWord` The lower word of the trigger time in picoseconds.

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
TLA700_E_SYSTEM_RUNNING	The operation cannot be performed when the system is acquiring data.
TLA700_E_OUT_OF_MEMORY	There is not enough memory to perform the operation.
TLA700_E_FAILED	The operation was unsuccessful.

Example

```
(Microsoft Visual Basic)

Dim App As Object
Dim Sys As Object
Dim DSO As Object
Dim Slot As Long
Dim TimeHigh As Long
Dim TimeLow As Long

Set App = CreateObject("TLA700.Application")
 'Get system.
Set Sys = App.GetSystem
 'Get module.
Set DSO = Sys.GetModuleBySlot(Slot)
 'Get trigger time.
DSO.GetTriggerTime TimeHigh, TimeLow
```

Remarks

Trigger time is a 64-bit value. This method returns the value in the form of two long values representing the high and low words. Though the low word is returned as a signed long value, it is to be treated as an *unsigned* value.

Trigger time is equivalent to the timestamp of the trigger sample in the acquisition buffer. Its value is the number of pico-seconds that have elapsed since the start of the acquisition. The start of the acquisition is the moment at which the system is enabled to begin acquiring data, however, there will always be some elapsed time before actual data acquisition begins. Other factors such as acquisitions buffer depth, clock rate, storage qualification of delay while waiting for a trigger condition to occur can also lengthen the period of time between the start of the acquisition and the time of the trigger sample in the storage buffer.

The Trigger time value is reported in terms of pico-seconds, and has been adjusted to include the frame offset and user specified time alignment values, if any. If additional, user defined, per-channel offsets are specified, those offsets are not reflected in the Trigger time value returned by this method. For a further explanation of these values, see the discussion of Timestamp values in the DataFormat section near the end of this document.

IDSOModule::GetTriggerSample

Description

This method returns the sample number of the trigger sample of the module.

ODL Syntax

```
HRESULT GetTriggerSample( [out, retval] long* pTriggerSample )
```

Arguments

`pTriggerSample` The sample number of the trigger sample of this module. Samples are numbered from 0.

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
TLA700_E_SYSTEM_RUNNING	The operation cannot be performed when the system is acquiring data.
TLA700_E_NO_TRIGGER_SAMPLE	There is no trigger sample.
TLA700_E_OUT_OF_MEMORY	There is not enough memory to perform the operation.
TLA700_E_FAILED	The operation was unsuccessful.

Example

```
(Microsoft Visual Basic)

Dim App As Object
Dim Sys As Object
Dim DSO As Object
Dim Slot As Long
Dim TriggerSample As Long

Set App = CreateObject("TLA700.Application")
 'Get system.
Set Sys = App.GetSystem
 'Get module.
Set DSO = Sys.GetModuleBySlot(Slot)
 'Get trigger sample.
TriggerSample = DSO.GetTriggerSample
```

IDSOModule::LoadModule

Description

This method loads a module from the specified TLA system or module file onto the current module.

ODL Syntax

```
HRESULT  LoadModule(  [in]  BSTR  ModulePath,
                      [in]  BSTR  ModuleName  )
```

Arguments

`ModulePath` The full path to the specified TLA system or module file. For example: **"C:\My Documents\My System.tla"**

`ModuleName` The name of the module in the specified file to load.

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
TLA700_E_LOAD_INVALID_FILE	An error occurred opening the file for reading.
TLA700_E_LOAD_INVALID_MODULE_NAME	Invalid “ModuleName” argument.
TLA700_E_LOAD_INVALID_MODULE_TYPE	The module specified is not compatible.
TLA700_E_LOAD_NOT_ENOUGH_CHANNELS	There are not enough channels to load the specified module.
TLA700_E_LOAD_MODULE_ERROR	An error occurred loading a module from the file.
TLA700_E_SYSTEM_RUNNING	The operation cannot be performed when the system is acquiring data.
TLA700_E_OUT_OF_MEMORY	There is not enough memory to perform the operation.
TLA700_E_FAILED	The operation was unsuccessful.
TLA700_E_MODAL_DIALOG_OPEN	The operation was disallowed because a modal dialog is open.

Example

```
(Microsoft Visual Basic)

Dim App As Object
Dim Sys As Object
Dim DSO As Object
Dim Slot As Long

Set App = CreateObject("TLA700.Application")
 ‘Get system.
Set Sys = App.GetSystem
 ‘Get module.
Set DSO = Sys.GetModuleBySlot(Slot)
 ‘Load module.
DSO.LoadModule "C:\My Documents\System1.tla", "DSO 1"
```

Remarks

All file paths without machine qualifiers refer to drives mapped on the TLA instrument.

Under some circumstances, when this operation is performed under the TLA Application Graphical User Interface, the user will be prompted to approve changes to some trigger actions. The purpose of the changes (if any) is to adjust individual module trigger programs for compatibility with System Trigger specification. When this operation is performed using the programmatic interface (TPI), no user prompt is possible and restored trigger programs will not be altered.

IDSOModule::Name

Description

This property allows the client to retrieve or set the name of the logical module.

ODL Syntax

```
[
   propget
]
HRESULT Name( [out, retval] BSTR* pModuleName )

[
   propput
]
HRESULT Name( [in] BSTR ModuleName )
```

Arguments (propget)

`pModuleName` - The name of the logical module.

Arguments (propput):

`ModuleName` - The name of the logical module.

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
TLA700_E_INVALID_MODULE_NAME	Invalid “ModuleName” argument.
TLA700_E_DUPLICATE_MODULE_NAME	This module name is already in use.
TLA700_E_OUT_OF_MEMORY	There is not enough memory to perform the operation.
TLA700_E_FAILED	The operation was unsuccessful.
TLA700_E_MODAL_DIALOG_OPEN	The operation was disallowed because a modal dialog is open.

Example

```
(Microsoft Visual Basic)

Dim App As Object
Dim Sys As Object
Dim DSO As Object
Dim ModuleName As String

Set App = CreateObject("TLA700.Application")
 ‘Get system.
Set Sys = App.GetSystem
 ‘Get module in slot 3.
Set DSO = Sys.GetModuleBySlot(3)
 ‘Get module name.
ModuleName = DSO.Name
 ‘Set module name.
DSO.Name = "My DSO"
```

Remarks

When retrieving the name, the TLA server will allocate the space for the returned string. The client is responsible for freeing it when it is no longer in use.

A module name cannot exceed twelve characters in length and must contain at least one non-whitespace character. Modules must be assigned unique names.

IDSOModule::SaveModule

Description

This method saves the module setup and, optionally, the acquisition data to a file.

ODL Syntax

```
HRESULT  SaveModule(  [in]  BSTR  ModulePath,
                      [in]  BSTR  UserComment,
                      [in]  long  SaveData  )
```

Arguments

`ModulePath` The full path to the destination file.For example:

"C:\My Documents\My Module.tla"

`UserComment` The user comment to be saved in the file.

`SaveData` This flag takes one of the values from the following table:

Value	Description
TLA700_SAVE_NO_DATA (0)	Do not save acquisition data in file.
TLA700_SAVE_DATA (1)	Save acquisition data in file.

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
TLA700_E_INVALID_SAVE_DATA	Invalid “SaveData” argument.
TLA700_E_SAVE_INVALID_FILE	An error occurred opening this file for writing.
TLA700_E_SAVE_OUT_OF_SPACE	There is not enough disk space to perform this operation.
TLA700_E_SAVE_ERROR	An error occurred during the save operation.
TLA700_E_SYSTEM_RUNNING	The operation cannot be performed when the system is acquiring data.
TLA700_E_OUT_OF_MEMORY	There is not enough memory to perform the operation.
TLA700_E_SAVE_FILE_SIZE_LIMIT_EXCEEDED	The save operation cannot be performed because the size of the saved file will exceed the maximum limit supported by the file system.
TLA700_E_FAILED	The operation was unsuccessful.

Example

```
(Microsoft Visual Basic)

Dim App As Object
Dim Sys As Object
Dim DSO As Object
Dim Slot As Long
```

```
Set App = CreateObject("TLA700.Application")
 'Get system.
Set Sys = App.GetSystem
 'Get module.
Set DSO = Sys.GetModuleBySlot(Slot)
 'Save module.
DSO.SaveModule "C:\My Documents\a.tla", "My module", 1
```

Remarks

All file paths without machine qualifiers refer to drives mapped on the TLA instrument.

If the file already exists, it will be overwritten.

ILAModule::DefineDataFormat

Description

This method is used to specify the format of the data to be returned in subsequent queries for acquisition data (using ILAModule::GetData() and ILAModule::ExportData()).

ODL Syntax

```
HRESULT  DefineDataFormat(  [in]  long  DataSet,
                            [in]  BSTR  Components,
                            [in]  long  DataType,
                            [out,  retval]  long*  pBytesPerSample  )
```

Arguments

`DataSet` This takes one of the following values:

Value	Description
TLA700_MAIN_DATASET (0)	Main DataSet
TLA700_MAGNIVU_DATASET (1)	MagniVu DataSet
TLA700_VIOLATION_DATASET (2)	Glitch or Setup and Hold DataSet

`Components` This takes one of the following forms:

- `"RawWithTimestamp"` Raw samples, ungrouped, including time stamp data.
- `"RawWithoutTimestamp"` Raw samples, ungrouped, without time stamp data.

Note: Channel polarity for the following grouped formats is used as specified in the LA Setup Window.

`"AllGroupsWithTimestamp"` Data is returned grouped according to the current channel grouping. Data for all channel groups is returned in the order in which groups are defined in the LA Setup window. Time stamp data is also returned.

`"AllGroupsWithoutTimestamp"` Data is returned grouped according to the current channel grouping. Time stamp data is not returned. Data for all channel groups is returned in the order in which groups are defined in the LA Setup Window.

`"GroupList:<grp>,<grp>,<grp>,Timestamp"` Specifies exactly which channel groups to return and the order in which they are to be returned. Groups are specified using their user names as shown in the LA Setup window. The same group can be included more than once. There is no way to get a "Sample number" group. Do not use any extra white space when specifying the group list. To get time stamp data, include "Timestamp" in the list.

For details on the format of these components, refer to the following topics:

Binary data formats for LA modules

Text data formats for LA modules

When acquisition data is returned in a raw format, time stamp values are returned in units of "ticks" where each tick represents a fixed amount of time. To obtain the time stamp values in picoseconds, it must be adjusted as follows:

Time stamp *value in ps = (Time stamp value in ticks *Time stamp Multiplier) +Expansion mainframe Offset +User-defined time alignment in ps*

The Time stamp Multiplier is obtained by calling ILAModule::GetTimestampMultiplier(). Expansion Mainframe Offset is 36000 ps if the module is in an expansion mainframe or 0 ps if it is not. User-defined time alignment is any time alignment defined for the module by the user.

When acquisition data is returned in a grouped format, time stamp values are returned in picoseconds, using the above calculation such that frame offset and user offset, if any are incorporated into the value.

`DataType` This takes one of the values from the following table:

Value	Description
TLA700_BINARY (0)	Binary
TLA700_TEXT_SPACE (1)	ASCII, space delimiter
TLA700_TEXT_TAB (2)	ASCII, tab delimiter
TLA700_TEXT_COMMA (3)	ASCII, comma delimiter

`pBytesPerSample` The number of bytes required per sample. The value returned is for the data format specified in this call. This value is meaningful only for binary data formats. For text data formats, the value returned is –1.

Refer to ILAModule::GetBytesPerSample() for details.

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
TLA700_E_INVALID_DATASET	Invalid “DataSet” argument.
TLA700_E_INVALID_COMPONENTS	Invalid “Components” argument.
TLA700_E_INVALID_DATA_TYPE	Invalid “DataType” argument.
TLA700_E_OUT_OF_MEMORY	There is not enough memory to perform the operation.
TLA700_E_FAILED	The operation was unsuccessful.

Example

```
(Microsoft Visual Basic)

Dim App As Object
Dim Sys As Object
Dim LA As Object
Dim Slot As Long
Dim BytesPerSample As Long

Set App = CreateObject("TLA700.Application")
 'Get system.
Set Sys = App.GetSystem
 'Get module in slot 3.
```

```
Set LA = Sys.GetModuleBySlot(3)
 'Get main data in binary grouped format.
BytesPerSample = LA.DefineDataFormat(0,"GroupList:A2,A3",0)
```

Remarks

Remember to invoke ILAModule::DefineDataFormat() after making calls to set up your module. Operations, such as ILAModule::LoadModule(), can change the setup of the module and may invalidate a previously defined data format.

If a grouped format is specified, groups that are empty are ignored. For grouped formats, an error is returned if the total amount of grouped data (excluding time stamp) is not greater than 0 bytes.

Because the comma is used to delimit groups in the "GroupList:…" form of the Components argument, a group name cannot contain a comma. Since the string Timestamp is used to retrieve time stamp values in the same context, groups cannot be named Timestamp.

The Timestamp group cannot be specified with the MagniVu data set.

See Tips for improving LA data transfer performance for information about optimizing the acquisition data transfer.

ILAModule::DeleteChannelGroup

Description

This method is used to delete a channel group.

ODL Syntax

```
HRESULT DeleteChannelGroup( [in] BSTR UserChannelGroupName )
```

Arguments

`UserChannelGroupName` – This is the name of the current group to be deleted.

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
TLA700_E_INVALID_USER_CHANNEL_GROUP_NAME	Invalid “UserChannelGroupName”.
TLA700_E_OUT_OF_MEMORY	There is not enough memory to perform the operation.
TLA700_E_MODAL_DIALOG_OPEN	The operation was disallowed because a modal dialog is open.
TLA700_E_FAILED	The operation was unsuccessful.

Example

```
 (Microsoft Visual Basic)
Dim App As Object
Dim Sys As Object
Dim LA As Object

Set App = CreateObject("TLA700.Application")

'Get system.
Set Sys = App.GetSystem
      …
'Get module in slot 3.
Set LA = Sys.GetModuleBySlot(3)

'Delete the channel group named Address
LA.DeleteChannelGroup "Address"
```

ILAModule::Enabled

Description

This property allows the client to enable/disable the logical module.

ODL Syntax

```
[
   propget
]
HRESULT Enabled( [out, retval] VARIANT_BOOL* pEnabled )

[
   propput
]
HRESULT Enabled( [in] VARIANT_BOOL Enabled )
```

Arguments (propget)

`pEnabled` - The enabled status of the logical module.

Arguments (propput):

`Enabled` - The enabled status of the logical module.

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
TLA700_E_SYSTEM_RUNNING	The operation cannot be performed when the system is acquiring data.
TLA700_E_OUT_OF_MEMORY	There is not enough memory to perform the operation.
TLA700_E_FAILED	The operation was unsuccessful.
TLA700_E_MODAL_DIALOG_OPEN	The operation was disallowed because a modal dialog box is open.

Example

```
(Microsoft Visual Basic)

Dim App As Object
Dim Sys As Object
Dim LA As Object
Dim Enabled As Boolean

Set App = CreateObject("TLA700.Application")
'Get system.
Set Sys = App.GetSystem
 'Get module in slot 3.
Set LA = Sys.GetModuleBySlot(3)
 'Get enabled status.
Enabled = LA.Enabled
'Enable module.
LA.Enabled = True
```

ILAModule::ExportData

Description

This method exports the specified acquisition data to a file.

ODL Syntax

```
HRESULT  ExportData(  [in]  long  FirstSample,
                      [in]  long  NumSamples,
                      [in]  BSTR  DataPath,
                      [in]  long  Mode  )
```

Arguments

`FirstSample` The sample number of the first sample to export.

`NumSamples` The number of samples to export.

`DataPath` The complete path to the destination file. For example:

"C:\My Documents\My Data.txt" for text format
"C:\My Documents\My Data.tbf" for binary format

`Mode` This argument takes one of the following values:

Value	Description
TLA700_APPEND (0)	Causes the new data to be appended to the existing contents of the file.
TLA700_OVERWRITE (1)	Causes the file to be overwritten.

The composition and format of the data must be specified by the ILAModule::DefineDataFormat() call.

If the data type is TLA700_BINARY, the file is a stream of bytes, each sample consisting of ILAModule::GetBytesPerSample() bytes.

If the data type is TLA700_TEXT_*, the file consists of a stream of newline-separated strings, one string per sample.

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
TLA700_E_INVALID_DATA_FORMAT	A valid data format must be defined before calling this method. The data format you defined previously may no longer be valid.
TLA700_E_VALUE_NOT_AVAILABLE	The required value(s) are not available.
TLA700_E_INVALID_FIRST_SAMPLE	Invalid “FirstSample” argument.
TLA700_E_INVALID_NUM_SAMPLES	Invalid “NumSamples” argument.
TLA700_E_INVALID_MODE	Invalid “Mode” argument.
TLA700_E_EXPORT_INVALID_FILE	An error occurred opening this file for writing.
TLA700_E_EXPORT_ERROR	An error occurred during the export operation.
TLA700_E_OUT_OF_MEMORY	There is not enough memory to perform the operation.
TLA700_E_FAILED	The operation was unsuccessful.

Example

```
(Microsoft Visual Basic)

Dim App As Object
```

```
Dim Sys As Object
Dim LA As Object
Dim BytesPerSample As Long

Set App = CreateObject("TLA700.Application")
‘Get system.
Set Sys = App.GetSystem
 ‘Get module in slot 3.
Set LA = Sys.GetModuleBySlot(3)
 ‘Export data in text format.
BytesPerSample = LA.DefineDataFormat(0,"GroupList:A2,A3",1)
LA.ExportData 0,10,"C:\export.txt",0
```

Remarks

The value specified for FirstSample must be greater than or equal to zero and less than the total number of samples acquired. NumSamples must be greater than zero, and FirstSample + NumSamples must be less than or equal to the total number of samples. If these conditions are not met, an error code is returned.

When sample suppression is not being used:

(FirstSample >= 0) and (FirstSample < Total # of acquired samples)

(NumSamples > 0) and ((FirstSample + NumSamples) <= Total # of acquired samples)

When sample suppression is being used:

(FirstSample >= 0) and (FirstSample < Total # of unsuppressed samples)

(NumSamples > 0) and ((FirstSample+NumSamples) <= Total # of unsuppressed samples)

A data format must be defined using ILAModule::DefineDataFormat() before calling this method.

All file paths without machine qualifiers refer to drives mapped on the TLA instrument.

ILAModule::GetBeginTime

Description

This method returns the begin time of the module for the specified data set.

ODL Syntax

```
HRESULT  GetBeginTime(  [in]  long  DataSet,
                        [out]  long*  pTimeHighWord,
                        [out]  long*  pTimeLowWord  )
```

Arguments

`DataSet` This takes one of the values from the following table:

Value	Description
TLA700_MAIN_DATASET (0)	Main DataSet
TLA700_MAGNIVU_DATASET (1)	MagniVu DataSet
TLA700_VIOLATION_DATASET (2)	Glitch or Setup and Hold DataSet

`pTimeHighWord` The higher word of the begin time in picoseconds.

`pTimeLowWord` The lower word of the begin time in picoseconds.

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
TLA700_E_INVALID_DATASET	Invalid “DataSet” argument.
TLA700_E_SYSTEM_RUNNING	The operation cannot be performed when the system is acquiring data.
TLA700_E_NO_SAMPLES	There are no data samples.
TLA700_E_OUT_OF_MEMORY	There is not enough memory to perform the operation.
TLA700_E_FAILED	The operation was unsuccessful.

Example

```
(Microsoft Visual Basic)

Dim App As Object
Dim Sys As Object
Dim LA As Object
Dim TimeHigh As Long
Dim TimeLow As Long
Dim DataSet as Long

Set App = CreateObject("TLA700.Application")
 'Get system.
Set Sys = App.GetSystem
 'Get module in slot 3.
Set LA = Sys.GetModuleBySlot(3)
 'Get begin time.
DataSet = 0
LA.GetBeginTime DataSet,TimeHigh,TimeLow
```

Remarks

The values returned for TLA700_MAIN_DATASET and TLA700_VIOLATION_DATASET are the same, provided that glitches or setup and hold data were acquired.

Begin time is a 64-bit value. This method returns the value in the form of two long values representing the high and low words. Though the low word is returned in a signed long value, it is to be treated as an *unsigned* value.

Begin time is equivalent to the timestamp of the first sample in the acquisition buffer. Its value is the number of pico-seconds that have elapsed since the start of the acquisition. The start of the acquisition is the moment at which the system is enabled to begin acquiring data, however, there will always be some elapsed time before actual data acquisition begins. Other factors such as acquisition buffer depth, clock rate, storage qualification or delay while waiting for a trigger condition to occur can also affect the period of time between the start of the acquisition and the time of the first sample in the storage buffer.

The Begin time value is reported in terms of pico-seconds, and has been adjusted to include the frame offset and user specified time alignment values, if any. For a further explanation of these values, see the discussion of Timestamp values in the DataFormat section near the end of this document.

ILAModule::GetBytesPerSample

Description

This method returns the number of bytes required for each sample of data when a binary data format is specified.

ODL Syntax:

```
HRESULT GetBytesPerSample( [out, retval] long* pBytesPerSample )
```

Arguments

`pBytesPerSample` The number of bytes required per sample. This value is for the current data format that was specified in the most recent call to ILAModule::DefineDataFormat(). This value is meaningful only if a binary data format was specified. If a text data format was specified, the value returned is –1.

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
TLA700_E_INVALID_DATA_FORMAT	A valid data format must be defined before calling this method. The data format you defined previously may no longer be valid.
TLA700_E_OUT_OF_MEMORY	There is not enough memory to perform the operation.
TLA700_E_FAILED	The operation was unsuccessful.

Example

```
(Microsoft Visual Basic)

Dim App As Object
Dim Sys As Object
Dim LA As Object
Dim BytesPerSample As Long

Set App = CreateObject("TLA700.Application")
 'Get system.
Set Sys = App.GetSystem
 'Get module in slot 3.
Set LA = Sys.GetModuleBySlot(3)
BytesPerSample = LA.DefineDataFormat(0,"GroupList:A2,A3",0)
BytesPerSample = LA.GetBytesPerSample
```

Remarks

A data format must be defined using ILAModule::DefineDataFormat() before calling this method. The value returned is meaningful only for binary data formats.

ILAModule::GetChannelGroup

Description

This method is used to get of channels assigned to a channel group.

ODL Syntax

```
HRESULT GetChannelGroup( [in] BSTR UserChannelGroupName,
                         [out, retval] BSTR* ChannelNameList )
```

Arguments

UserChannelGroupName	This is the user defined name of the channel group whose channel list is to be retrieved
ChannelNameList	This is the list of channel names assigned to the channel group. For individual channels the syntax is the hardware pod name followed by the channel number enclosed in parentheses, (e.g. A0(1), A0(2), etc.). Groups of contiguous channels can be specified using a similar syntax' using a range of channel numbers within the parentheses, (e.g. A0(7-0), A1(3-5), etc.). When all of the channels in a hardware pod are to be specified, a shorthand notation is allowed using just an empty pair of parentheses, (e.g. A0(), A1(), etc.). The syntax for clock and qualifier channels is the type identifier "CK" or "Q" respectively, followed by a number, (e.g. CK0, CK1, Q0, Q1, etc.). Multiple channels or channel groups can be specified using a comma separated list. Embedded spaces are not allowed.

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
TLA700_E_INVALID_USER_CHANNEL_GROUP_NAME	Invalid "UserChannelGroupName" argument.
TLA700_E_OUT_OF_MEMORY	There is not enough memory to perform the operation.
TLA700_E_FAILED	The operation was unsuccessful.

Example

```
 (Microsoft Visual Basic)

Dim App As Object
Dim Sys As Object
Dim LA As Object
Dim ChannelList As String

Set App = CreateObject("TLA700.Application")

'Get system.
Set Sys = App.GetSystem
      …
'Get module in slot 3.
Set LA = Sys.GetModuleBySlot(3)
```

```
'Get the channel list assigned to group "Address"
ChannelList = LA.GetChannelGroup("Address")
```

ILAModule::GetChannelName

Description

This method is used to get the user assigned name for a channel of this module.

ODL Syntax

```
HRESULT GetChannelName( [in] BSTR HWChannelName
                                                    [out, retval] BSTR*
UserChannelName )
```

Arguments

`HWChannelName`	This is the hardware name of the channel whose user name is to be retrieved. For normal acquisition channels, the syntax is the hardware pod name followed by the channel number enclosed in parentheses, (e.g. A0(1), A0(2) etc.). The syntax for clock and qualifier channels is the type identifier "CK" or "Q" respectively, followed by a number, (e.g. CK0, CK1, Q0, Q1, etc.).
`UserChannelName`	The user name assigned to the channel

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
TLA700_E_INVALID_HW_CHANNEL_NAME	Invalid “HWChannelName” argument.
TLA700_E_OUT_OF_MEMORY	There is not enough memory to perform the operation.
TLA700_E_FAILED	The operation was unsuccessful.

Example

(Microsoft Visual Basic)

```
Dim App As Object
Dim Sys As Object
Dim LA As Object
Dim Name As String

Set App = CreateObject("TLA700.Application")

'Get system.
Set Sys = App.GetSystem
        ...
'Get module in slot 3.
Set LA = Sys.GetModuleBySlot(3)

'Get User channel name for probe C3, channel 4
Name = LA.GetChannelName("C3(4)")
```

ILAModule::GetCounterValue

Description

This method returns the final value of a module counter from the last acquisition.

ODL Syntax

```
HRESULT  GetCounterValue(  [in]  long  CounterID,
                           [out]  long*  pCounterHighWord,
                           [out]  long*  pCounterLowWord  )
```

Arguments

`CounterID`	Identifies the counter. It is either 1 or 2.
`pCounterHighWord`	The higher word of the end value of the specified counter.
`pCounterLowWord`	The lower word of the end value of the specified counter.

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
TLA700_E_INVALID_COUNTER_ID	Invalid “CounterID” argument.
TLA700_E_VALUE_NOT_AVAILABLE	The required value(s) are not available.
TLA700_E_COUNTER_NOT_USED	The counter is not currently being used.
TLA700_E_OUT_OF_MEMORY	There is not enough memory to perform the operation.
TLA700_E_FAILED	The operation was unsuccessful.

Example

(Microsoft Visual Basic)

```
Dim App As Object
Dim Sys As Object
Dim LA As Object
Dim Slot As Long
Dim CounterHigh As Long
Dim CounterLow As Long

Set App = CreateObject("TLA700.Application")
 'Get system.
Set Sys = App.GetSystem
 'Get module in slot 3.
Set LA = Sys.GetModuleBySlot(3)
 'Get counter value.
LA.GetCounterValue 1,CounterHigh,CounterLow
```

Remarks

A counter value is a 64-bit value. This method returns the value in the form of two 32-bit long values representing the high and low words. Although the low word is returned as a signed long value, treat it as an unsigned value.

Counter values are not available when the TLA server is running.

All TLA600 Series Logic Analyzers and all logic analyzer modules use 64 bits for the counter, however the counter values are represented differently. For TLA600 Series Logic Analyzers and TLA7Lx/Mx/Nx/Px/Qx Series Logic Analyzer modules, the 64 bits represent an unsigned value, ranging from 0 to (2^64)-1. The TLA7Axx Series Logic Analyzer modules have counters that can be decremented and the counter values can be negative. As a result, the 64 counter bits are a two's complement representation of a signed value ranging from –(2^63) to +(2^63)-1.

To interpret the counter value correctly, a TPI client must know whether the value is signed or unsigned. The counter representation depends on the logic analyzer module type. Use the ISystem::GetModulePropertiesBySlot method to determine the logic analyzer module type. If the module is a TLA7Axx Series Logic Analyzer module, then the 64 bit counter is signed; otherwise, the counter is unsigned.

ILAModule::GetData

Description

This method returns the specified acquisition data.

ODL Syntax

```
HRESULT  GetData(  [in]  long  FirstSample,
                   [in]  long  NumSamples,
                   [out,  retval]  VARIANT*  pData  )
```

Arguments

`FirstSample` The sample number of the first sample to return. Sample numbers start with 0.

`NumSamples` The total number of samples to return.

`pData`

The acquisition data. The composition and format of the data is determined by the ILAModule::DefineDataFormat() call. Data is returned as a VARIANT. The variant is of type VT_ARRAY and points to a SAFEARRAYThe SAFEARRAY has a dimension of one and has NumSamples entries.

If the data type is TLA700_BINARY, the SAFEARRAY is of element type VT_UI1 (unsigned char). The size of the array is: NumSamples * ILAModule::GetBytesPerSample().

If the data type is TLA700_TEXT_*, the element type is VT_BSTR (string). The size of the array is NumSamples.

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
TLA700_E_INVALID_COUNTER_ID	Invalid "CounterID" argument.
TLA700_E_ VALUE_NOT AVAILABLE	The required value(s) are not available.
TLA700_E_COUNTER_NOT_USED	The counter is not currently being used.
TLA700_E_OUT_OF_MEMORY	There is not enough memory to perform the operation.
TLA700_E_BUFFER_LIMIT_EXCEEDED	The size of the requested data buffer exceeded the maximum limit.
TLA700_E_FAILED	The operation was unsuccessful.

Example

```
(Microsoft Visual Basic)

Dim App As Object
Dim Sys As Object
Dim LA As Object
Dim BytesPerSample As Long
Dim NumSamples As Long
Dim DataArray As Variant

Set App = CreateObject("TLA700.Application")
 'Get system.
Set Sys = App.GetSystem
 'Get module in slot 3.
Set LA = Sys.GetModuleBySlot(3)
 'Get main data in binary grouped format.
BytesPerSample = LA.DefineDataFormat(0,"GroupList:A2,A3",0)
DataArray = LA.GetData(0,NumSamples)
 'Process samples.
For I = 0 To NumSamples * NumBytesPerSample –
      NumBytesPerSample Step NumBytesPerSample
'Process the J bytes in Ith sample.
      For J = 0 To NumBytesPerSample - 1
                  'Process DataArray(I + J)
      Next J
Next I
```

Remarks

The value specified for FirstSample must be greater than or equal to zero and less than the total number of samples. NumSamples must be greater than zero and FirstSample + NumSamples must be less than or equal to the total number of samples. This implies the following:

When sample suppression is not being used:

(FirstSample >= 0) and (FirstSample < Total # of acquired samples)

(NumSamples > 0) and ((FirstSample + NumSamples) <= Total # of acquired samples)

When sample suppression is being used:

(FirstSample >= 0) and (FirstSample < Total # of unsuppressed samples)

(NumSamples > 0) and ((FirstSample+NumSamples) <= Total # of unsuppressed samples)

A data format must be defined using ILAModule::DefineDataFormat () before calling this method. The TLA server will allocate the space for the array of data. The client is responsible for freeing it when it is no longer in use.

For information on important performance considerations see Tips for Improving Data Transfer Performance.

When using Internal 2X, Internal 4X, External 2X, or External 4X clocking modes, the acquired data appears different than when using standard Internal or External clocking modes. For example when using Internal clocking, External clocking, or Source Synchronous clocking, one data sample in the listing window equates to one data sample in the acquisition memory. However if you use Internal 2X clocking or External 2X clocking, two data samples in the Listing window equate to one data sample in the acquisition memory. If you use Internal 4X clocking or External 4X clocking, four data samples in the Listing window equate to one data sample in acquisition memory. However, in both cases, the Listing window displays dashes for duplicate demultiplex group values while the data returned by TPI shows the actual values. For more information, see Internal 2X clocking mode, Internal 4X clocking mode, External 2X clocking mode, External 4X clocking mode in the Appendix, or refer to the TLA application online help.

ILAModule::GetEndTime

Description

This method returns the end time of the module for the specified data set.

ODL Syntax

```
HRESULT  GetEndTime(  [in]  long  DataSet,
                      [out]  long*  pTimeHighWord,
                      [out]  long*  pTimeLowWord  )
```

Arguments

`DataSet` This takes one of the values from the following table:

Value	Description
TLA700_MAIN_DATASET (0)	Main DataSet
TLA700_MAGNIVU_DATASET (1)	MagniVu DataSet
TLA700_E_VIOLATION_DATASET (2)	Glitch or Setup and Hold DataSet

`pTimeHighWord` The higher word of the end time in picoseconds.

`pTimeLowWord` The lower word of the end time in picoseconds.

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
TLA700_E_INVALID_DATASET	Invalid “DataSet” argument.
TLA700_E_SYSTEM_RUNNING	The operation cannot be performed when the system is acquiring data.
TLA700_E_NO_SAMPLES	There are no data samples.
TLA700_E_OUT_OF_MEMORY	There is not enough memory to perform the operation.
TLA700_E_FAILED	The operation was unsuccessful.

Example

```
(Microsoft Visual Basic)

Dim App As Object
Dim Sys As Object
Dim LA As Object
Dim TimeHigh As Long
Dim TimeLow As Long
Dim DataSet as Long

Set App = CreateObject("TLA700.Application")
 ‘Get system.
Set Sys = App.GetSystem
‘Get module in slot 3.
Set LA = Sys.GetModuleBySlot(3)
 ‘Get end time.
DataSet = 0
LA.GetEndTime DataSet,TimeHigh,TimeLow
```

Remarks

The values returned for TLA700_MAIN_DATASET and TLA700_VIOLATION_DATASET are the same, provided that glitches or setup and hold data were acquired.

End time is a 64-bit value. This method returns the value in the form of two long values representing the high and low words. Though the low word is returned in a signed long value, it is to be treated as an *unsigned* value.

End time is equivalent to the timestamp of the last sample in the acquisition buffer. Its value is the number of pico-seconds that have elapsed since the start of the acquisition. The start of the acquisition is the moment at which the system is enabled to begin acquiring data, however, there will always be some elapsed time before actual data acquisition begins. Other factors such as acquisition buffer depth, clock rate, storage qualification or delay while waiting for a trigger condition to occur can also affect the period of time between the start of the acquisition and the time of the last sample in the storage buffer.

The End time value is reported in terms of pico-seconds, and has been adjusted to include the frame offset and user specified time alignment values, if any. For a further explanation of these values, see the discussion of Timestamp values in the DataFormat section near the end of this document.

ILAModule::GetGroupNames

Description

This method retrieves the names of all groups defined in the module setup.

ODL Syntax

```
HRESULT GetGroupNames( [out, retval] VARIANT* pGroupNames )
```

Arguments

`pGroupNames` – The group names.

Group names are returned as a VARIANT. The variant is of type VT_ARRAY and points to a SAFEARRAY. The SAFEARRAY has dimension 1 and the elements are of type VT_BSTR. The number of groups is equal to the number of elements in the SAFEARRAY. The groups are returned in the same order as they are specified in the LA Setup Window.

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
TLA700_E_OUT_OF_MEMORY	There is not enough memory to perform the operation.
TLA700_E_FAILED	The operation was unsuccessful.

Example

```
(Microsoft Visual Basic)

Dim App As Object
Dim Sys As Object
Dim LA As Object
Dim G As Variant
Dim Groups As Variant

Set App = CreateObject("TLA700.Application")
 'Get system.
Set Sys = App.GetSystem
 'Get module in slot 3.
Set LA = Sys.GetModuleBySlot(3)
 'Get group names.
Groups = LA.GetGroupNames
`Access group names.
For Each G In Groups
'Use group name in G.
Next G
```

Remarks

If there are no groups defined, the SAFEARRAY returned will be empty.

ILAModule::GetGroupSize

Description

This method retrieves the number of channels in a specified group defined in the module setup.

ODL Syntax

```
HRESULT GetGroupSize( BSTR GroupName, [out, retval] long* pGroupSize )
```

Arguments

`pGroupSize` – The number of channels in the specified group.

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
TLA700_E_INVALID_GROUP_NAME	Invalid “GroupName” argument.
TLA700_E_OUT_OF_MEMORY	There is not enough memory to perform the operation.
TLA700_E_FAILED	The operation was unsuccessful.

Example

```
(Microsoft Visual Basic)

Dim App As Object
Dim Sys As Object
Dim LA As Object
Dim G As Variant
Dim Groups As Variant
Dim GroupSize As Long

Set App = CreateObject("TLA700.Application")
 ‘Get system.
Set Sys = App.GetSystem
 ‘Get module in slot 3.
Set LA = Sys.GetModuleBySlot(3)
 ‘Get group names.
Groups = LA.GetGroupNames
`Access group sizes.
For Each G In Groups
    GroupSize = LA.GetGroupSize( G )
Next G
```

Remarks

This method returns the actual number of channels in the specified group. When data is transferred in binary via GetData()/ExportData(), channel groups are padded to the nearest byte boundary.

ILAModule::GetNumSamples

Description

This method returns the number of unsuppressed samples in the acquisition memory of the module for the specified data set.

ODL Syntax

```
HRESULT  GetNumSamples(  [in]  long  DataSet,
                         [out,  retval]  long*  pNumSamples  )
```

Arguments

`DataSet` This takes one of the values from the following table:

Value	Description
TLA700_MAIN_DATASET (0)	Main DataSet
TLA700_MAGNIVU_DATASET (1)	MagniVu DataSet
TLA700_VIOLATION_DATASET (2)	Glitch or Setup and Hold DataSet

`pNumSamples` The number of samples in the acquisition memory of this module.

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
TLA700_E_INVALID_DATASET	Invalid “DataSet” argument.
TLA700_E_SYSTEM_RUNNING	The operation cannot be performed when the system is acquiring data.
TLA700_E_OUT_OF_MEMORY	There is not enough memory to perform the operation.
TLA700_E_FAILED	The operation was unsuccessful.

Example

```
(Microsoft Visual Basic)

Dim App As Object
Dim Sys As Object
Dim LA As Object
Dim NumSamples As Long
Dim DataSet As Long

Set App = CreateObject("TLA700.Application")
 'Get system.
Set Sys = App.GetSystem
 'Get module in slot 3.
Set LA = Sys.GetModuleBySlot(3)
 'Get number of samples.
DataSet = 0
NumSamples = LA.GetNumSamples(DataSet)
```

Remarks

This method will return a value of zero if no data is available.

The values returned for TLA700_MAIN_DATASET and TLA700_VIOLATION_DATASET are the same, provided glitches or setup and hold violations were acquired.

If sample suppression is being used, the values returned for TLA700_MAIN_DATASET and TLA700_VIOLATION_DATASET reflect the number of samples that are not suppressed.

ILAModule::GetStartTime

Description

This method returns the start date and start time for the module's acquisition.

ODL Syntax

```
HRESULT GetStartTime( [out, retval] VARIANT* pDate )
```

Arguments

`pDate` The start date and start time of the module's acquisition. This is returned as a VARIANT. The variant is of type VT_DATE.

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
TLA700_E_SYSTEM_RUNNING	The operation cannot be performed when the system is acquiring data.
TLA700_E_VALUE_NOT_AVAILABLE	The required value(s) are not available.
TLA700_E_OUT_OF_MEMORY	There is not enough memory to perform the operation.
TLA700_E_FAILED	The operation was unsuccessful.

Example

```
(Microsoft Visual Basic)

Dim App As Object
Dim Sys As Object
Dim LA As Object
Dim StartTime As Date

Set App = CreateObject("TLA700.Application")
 'Get system.
Set Sys = App.GetSystem
'Get module in slot 3.
Set LA = Sys.GetModuleBySlot(3)
 'Get start time.
StartTime = LA.GetStartTime
```

Remarks

The start of an acquisition is not the same as the first sample of data. The start of an acquisition is the moment at which the system is enabled to begin acquiring data. It takes some finite amount of time after the start of the acquisition before a module can actually store a sample of data.

ILAModule::GetTimerValue

Description

This method returns the final value of a module timer from the last acquisition.

ODL Syntax

```
HRESULT  GetTimerValue(  [in]  long  TimerID,
                         [out]  long*  pTimerHighWord,
                         [out]  long*  pTimerLowWord  )
```

Arguments

`TimerID` Identifies the timer. It is either 1 or 2.

`pTimerHighWord` The higher word of the end value of the specified timer in picoseconds.

`pTimerLowWord` The lower word of the end value of the specified timer in picoseconds.

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
TLA700_E_INVALID_TIMER_ID	Invalid “TimerID” argument.
TLA700_E_VALUE_NOT_AVAILABLE	The required value(s) are not available.
TLA700_E_TIMER_NOT_USED	The timer is not currently being used.
TLA700_E_OUT_OF_MEMORY	There is not enough memory to perform the operation.
TLA700_E_FAILED	The operation was unsuccessful.

Example

```
(Microsoft Visual Basic)

Dim App As Object
Dim Sys As Object
Dim LA As Object
Dim Slot As Long
Dim TimerHigh As Long
Dim TimerLow As Long

Set App = CreateObject("TLA700.Application")
 ‘Get system.
Set Sys = App.GetSystem
 ‘Get module in slot 3.
Set LA = Sys.GetModuleBySlot(3)
 ‘Get timer value.
LA.GetTimerValue 1,TimerHigh,TimerLow
```

Remarks

A timer value is a 64-bit value. This method returns the value in the form of two 32-bit long values representing the high and low words. Although the low word is returned as a signed long value, treat it as an unsigned value.

Timer values are not available when the TLA server is running.

ILAModule::GetTimestampMultiplier

Description

This multiplier is to be used only when acquisition data is returned in binary format. This method returns the value that must be multiplied with a timestamp value to obtain the final timestamp value in picoseconds. This multiplier is to be used only when data is returned in RawWithTimestamp format.

ODL Syntax

```
HRESULT GetTimestampMultiplier( [out, retval] long* pMultiplier )
```

Arguments

`pMultiplier` The time stamp multiplier.

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
TLA700_E_VALUE_NOT_AVAILABLE	The required value(s) are not available.
TLA700_E_OUT_OF_MEMORY	There is not enough memory to perform the operation.
TLA700_E_FAILED	The operation was unsuccessful.

Example

```
(Microsoft Visual Basic)

Dim App As Object
Dim Sys As Object
Dim LA As Object
Dim Multiplier As Long

Set App = CreateObject("TLA700.Application")
‘Get system.
Set Sys = App.GetSystem
‘Get module in slot 3.
Set LA = Sys.GetModuleBySlot(3)
 ‘Get time stamp multiplier.
Multiplier = LA.GetTimestampMultiplier
```

ILAModule::GetTriggerSample

Description

This method returns the sample number of the trigger sample of the module for the specified data set.

ODL Syntax:

```
HRESULT GetTriggerSample( [in] long DataSet,
                          [out, retval] long* pTriggerSample )
```

Arguments

`DataSet` This takes one of the following values:

Value	Description
TLA700_MAIN_DATASET (0)	Main DataSet
TLA700_MAGNIVU_DATASET (1)	MagniVu DataSet
TLA700_VIOLATION_DATASET (2)	Glitch or Setup and Hold DataSet

`pTriggerSample` The sample number of the trigger sample of this module. Samples are numbered from 0.

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
TLA700_E_INVALID_DATASET	Invalid “DataSet” argument.
TLA700_E_SYSTEM_RUNNING	The operation cannot be performed when the system is acquiring data.
TLA700_E_NO_TRIGGER_SAMPLE	There is no trigger sample.
TLA700_E_OUT_OF_MEMORY	There is not enough memory to perform the operation.
TLA700_E_FAILED	The operation was unsuccessful.

Example

```
(Microsoft Visual Basic)

Dim App As Object
Dim Sys As Object
Dim LA As Object
Dim TriggerSample As Long
Dim DataSet As Long

Set App = CreateObject("TLA700.Application")
 ‘Get system.
Set Sys = App.GetSystem
 ‘Get module in slot 3.
Set LA = Sys.GetModuleBySlot(3)
 ‘Get trigger sample.
DataSet = 0
TriggerSample = LA.GetTriggerSample(DataSet)
```

Remarks

The values returned for TLA700_MAIN_DATASET and TLA700_VIOLATION _DATASET are the same, provided glitches or setup and hold violations were acquired.

If sample suppression is being used and the actual trigger sample is suppressed, the position of the trigger is adjusted to an unsuppressed sample. In this case the values returned for TLA700_MAIN_DATASET and TLA700_VIOLATION_DATASET will reflect the adjusted position of the trigger relative to the unsuppressed samples.

ILAModule::GetTriggerTime

Description

This method returns the time stamp of the trigger sample of the module's acquired data for the specified data set.

ODL Syntax

```
HRESULT  GetTriggerTime(  [in]  long  DataSet,
                          [out]  long*  pTimeHighWord,
                     [out]  long*  pTimeLowWord  )
```

Arguments

`DataSet` This takes one of the values from the following table:

Value	Description
TLA700_MAIN_DATASET (0)	Main DataSet
TLA700_MAGNIVU_DATASET (1)	MagniVu DataSet
TLA700_VIOLATION_DATASET (2)	Glitch or Setup and Hold DataSet

`pTimeHighWord` The higher word of the trigger time in picoseconds.

`pTimeLowWord` The lower word of the trigger time in picoseconds.

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
TLA700_E_INVALID_DATASET	Invalid "DataSet" argument.
TLA700_E_SYSTEM_RUNNING	The operation cannot be performed when the system is acquiring data.
TLA700_E_NO_TRIGGER_SAMPLE	There is no trigger sample.
TLA700_E_OUT_OF_MEMORY	There is not enough memory to perform the operation.
TLA700_E_FAILED	The operation was unsuccessful.

Example

```
(Microsoft Visual Basic)

Dim App As Object
Dim Sys As Object
Dim LA As Object
Dim TimeHigh As Long
Dim TimeLow As Long
Dim DataSet as Long

Set App = CreateObject("TLA700.Application")
 'Get system.
Set Sys = App.GetSystem
 'Get module in slot 3.
Set LA = Sys.GetModuleBySlot(3)
 'Get trigger time.
DataSet = 0
```

```
LA.GetTriggerTime DataSet,TimeHigh,TimeLow
```

Remarks

The values returned for TLA700_MAIN_DATASET and TLA700_VIOLATION_DATASET are the same, provided that glitches or setup and hold data were acquired.

Trigger time is a 64-bit value. This method returns the value in the form of two long values representing the high and low words. Though the low word is returned in a signed long value, it is to be treated as an *unsigned* value.

Trigger time is equivalent to the timestamp of the trigger sample in the acquisition buffer. Its value is the number of pico-seconds that have elapsed since the start of the acquisition. The start of the acquisition is the moment at which the system is enabled to begin acquiring data, however, there will always be some elapsed time before actual data acquisition begins. Other factors such as acquisition buffer depth, clock rate, storage qualification or delay while waiting for a trigger condition to occur can also affect the period of time between the start of the acquisition and the time of the trigger sample in the storage buffer.

The Trigger time value is reported in terms of pico-seconds, and has been adjusted to include the frame offset and user specified time alignment values, if any. For a further explanation of these values, see the discussion of Timestamp values in the DataFormat section near the end of this document.

ILAModule::LoadModule

Description

This method loads a module from the specified TLA system or module file onto the current module.

ODL Syntax

```
HRESULT  LoadModule(  [in]  BSTR  ModulePath,
                      [in]  BSTR  ModuleName  )
```

Arguments

`ModulePath` The full path to the specified TLA system or module file. For example: **"C:\My Documents\My System.tla"**

`ModuleName` The name of the module in the specified file to load.

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
TLA700_E_LOAD_INVALID_FILE	An error occurred opening the file for reading.
TLA700_E_LOAD_INVALID_MODULE_NAME	Invalid “ModuleName” argument.
TLA700_E_LOAD_INVALID_MODULE_TYPE	The module specified is not compatible.
TLA700_E_LOAD_NOT_ENOUGH_CHANNELS	There are not enough channels to load the specified module.
TLA700_E_LOAD_NOT_ENOUGH_MODULES	There are not enough physical modules to load the specified module.
TLA700_E_LOAD_MODULE_ERROR	An error occurred loading a module from a file.
TLA700_E_SYSTEM_RUNNING	The operation cannot be performed when the system is acquiring data.
TLA700_E_OUT_OF_MEMORY	There is not enough memory to perform the operation.
TLA700_E_FAILED	The operation was unsuccessful.
TABLE_E_MODAL_DIALOG_OPEN	The operation was disallowed because a modal dialog box is open.

Example

```
(Microsoft Visual Basic)

Dim App As Object
Dim Sys As Object
Dim LA As Object

Set App = CreateObject("TLA700.Application")
 'Get system.
Set Sys = App.GetSystem
 'Get module in slot 3.
Set LA = Sys.GetModuleBySlot(3)
 'Load module.
LA.LoadModule "C:\My Documents\System1.tla", "LA 1"
```

Remarks

Invoking this method does not result in a merge operation even if the destination module does not have enough channels/physical modules.

All file paths without machine qualifiers refer to drives mapped on the TLA instrument.

Under some circumstances, when this operation is performed under the TLA Application Graphical User Interface, the user will be prompted to approve changes to some trigger actions. The purpose of the proposed changes (if any) is to adjust individual module trigger programs for compatibility with the System Trigger specification. When this operation is performed using the programmatic (TPI), no user prompt is possible and restored trigger programs will not be altered.

ILAModule::LoadTrigger

Description

This method loads the trigger settings from the specified saved system or module file onto the current module.

ODL Syntax

```
HRESULT  LoadTrigger(  [in]  BSTR  ModulePath,
                       [in]  BSTR  ModuleName  )
```

Arguments

`ModulePath` The full path to the saved system or module file.
For example: **"C:\My Documents\My System.tla"**

`ModuleName` The name of the module in the specified file whose trigger settings are to be loaded.

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
TLA700_E_LOAD_INVALID_FILE	An error occurred opening the file for reading.
TLA700_E_LOAD_INVALID_MODULE_NAME	Invalid “ModuleName” argument.
TLA700_E_LOAD_INVALID_MODULE_TYPE	The module specified is not compatible.
TLA700_E_LOAD_MODULE_ERROR	An error occurred loading a module from the file.
TLA700_E_SYSTEM_RUNNING	The operation cannot be performed when the system is acquiring data.
TLA700_E_OUT_OF_MEMORY	There is not enough memory to perform the operation.
TLA700_E_FAILED	The operation was unsuccessful.
TLA700_E_MODAL_DIALOG_OPEN	The operation is disallowed because a modal dialog is open.

Example

```
(Microsoft Visual Basic)

Dim App As Object
Dim Sys As Object
Dim LA As Object

Set App = CreateObject("TLA700.Application")
 ‘Get system.
Set Sys = App.GetSystem
 ‘Get module in slot 3.
Set LA = Sys.GetModuleBySlot(3)
 ‘Load trigger.
LA.LoadTrigger "C:\My Documents\System1.tla", "LA 1"
```

Remarks

All file paths without machine qualifiers refer to drives mapped on the TLA instrument.

Under some circumstances, when this operation is performed under the TLA Application Graphical User Interface, the user will be prompted to approve changes to some trigger actions. The purpose of the proposed changes (if any) is to adjust individual module trigger programs for compatibility with the System Trigger specification. When this operation is performed using the programmatic (TPI), no user prompt is possible and restored trigger programs will not be altered.

ILAModule::MemoryDepth

Description

This property is used to get or set the Memory Depth for this module.

ODL Syntax

```
[
propget
]
HRESULT MemoryDepth( [out, retval] long* pMemoryDepth )

[
 propput
]
HRESULT MemoryDepth ( [in] long MemoryDepth )
```

Arguments (propget):

`pMemoryDepth` The current memory depth setting for the module.

Arguments (propput):

`MemoryDepth` The memory depth to be set for the module.

Value	Description
TLA700_MEMORY_DEPTH_128 (0)	128 samples
TLA700_MEMORY_DEPTH_256 (1)	256 samples
TLA700_MEMORY_DEPTH_512 (2)	512 samples
TLA700_MEMORY_DEPTH_1K (3)	1K samples
TLA700_MEMORY_DEPTH_2K (4)	2K samples
TLA700_MEMORY_DEPTH_4K (5)	4K samples
TLA700_MEMORY_DEPTH_8K (6)	8K samples
TLA700_MEMORY_DEPTH_16K (7)	16K samples
TLA700_MEMORY_DEPTH_32K (8)	32K samples
TLA700_MEMORY_DEPTH_64K (9)	64K samples
TLA700_MEMORY_DEPTH_128K (10)	128K samples
TLA700_MEMORY_DEPTH_256K (11)	256K samples
TLA700_MEMORY_DEPTH_512K (12)	512K samples
TLA700_MEMORY_DEPTH_1M (13)	1M samples
TLA700_MEMORY_DEPTH_2M (14)	2M samples
TLA700_MEMORY_DEPTH_4M (15)	4M samples
TLA700_MEMORY_DEPTH_8M (16)	8M samples
TLA700_MEMORY_DEPTH_16M (17)	16M samples
TLA700_MEMORY_DEPTH_32M (18)	32M samples
TLA700_MEMORY_DEPTH_64M (19)	64M samples
TLA700_MEMORY_DEPTH_128M (20)	128M samples
TLA700_MEMORY_DEPTH_256M (21)	256M samples
TLA700_MEMORY_DEPTH_512M (22)	512M samples
TLA700_MEMORY_DEPTH_1G (23)	1G samples

Return Code	Description
S_OK	The operation succeeded.
TLA700_E_INVALID_MEMORY_DEPTH	Invalid “MemoryDepth” argument.
TLA700_E_OUT_OF_MEMORY	There is not enough memory to perform the operation.
TLA700_E_MODAL_DIALOG_OPEN	The operation was disallowed because a modal dialog is open
TLA700_E_FAILED	The operation was unsuccessful.

Example

```
 (Microsoft Visual Basic)
Dim App As Object
Dim Sys As Object
Dim LA As Object
Dim Depth As Long

Set App = CreateObject("TLA700.Application")

'Get system.
Set Sys = App.GetSystem
      ...
 'Get module in slot 3.
Set LA = Sys.GetModuleBySlot(3)

'Get Memory Depth setting for this module.
Depth = LA.MemoryDepth

'Set Memory Depth to 1K samples
 LA.MemoryDepth = TLA700_MEMORY_DEPTH_1K
```

ILAModule::Name

Description

This property allows the client to retrieve or set the name of the logical module.

ODL Syntax

```
[
   propget
]
HRESULT Name( [out, retval] BSTR* pModuleName )

[
   propput
]
HRESULT Name( [in] BSTR ModuleName )
```

Arguments (propget)

`pModuleName` - The name of the logical module.

Arguments (propput):

`ModuleName` - The name of the logical module.

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
TLA700_E_INVALID_MODULE_NAME	Invalid “ModuleName” argument.
TLA700_E_DUPLICATE_MODULE_NAME	This module name is already in use.
TLA700_E_OUT_OF_MEMORY	There is not enough memory to perform the operation.
TLA700_E_FAILED	The operation was unsuccessful.
TLA700_E_MODAL_DIALOG_OPEN	The operation was disallowed because a modal dialog is open.

Example

```
(Microsoft Visual Basic)

Dim App As Object
Dim Sys As Object
Dim LA As Object
Dim ModuleName As String

Set App = CreateObject("TLA700.Application")
 'Get system.
Set Sys = App.GetSystem
 'Get module in slot 3.
Set LA = Sys.GetModuleBySlot(3)
 'Get module name.
ModuleName = LA.Name
 'Set module name.
LA.Name = "My LA"
```

Remarks

When retrieving the name, the TLA server will allocate the space for the returned string. The client is responsible for freeing it when it is no longer in use.

A module name cannot exceed twelve characters in length and must contain at least one non-whitespace character. Modules must be assigned unique names.

ILAModule::SaveModule

Description

This method saves the module to a file.

ODL Syntax

```
HRESULT  SaveModule(  [in]  BSTR  ModulePath,
                      [in]  BSTR  UserComment,
                      [in]  long  SaveData  )
```

Arguments

`ModulePath` The full path to the destination file.

For example: **"C:\My Documents\My Module.tla"**

`UserComment` The user comment to be saved in the file.

`SaveData` This flag takes one of the values from the following table:

Value	Description
TLA700_SAVE_NO_DATA (0)	Do not save acquisition data in file.
TLA700_SAVE_DATA (1)	Save acquisition data in file.
TLA700_SAVE_UNSUPPRESSED_DATA (2)	Save only unsuppressed acquisition data in file.

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
TLA700_E_INVALID_SAVE_DATA	Invalid "SaveData" argument.
TLA700_E_ SAVE_INVALID_FILE	An error occurred opening this file for writing.
TLA700_E_SAVE_OUT_OF_SPACE	There is not enough disk space to perform this operation.
TLA700_E_SAVE_ERROR	An error occurred during the save operation.
TLA700_E_SYSTEM_RUNNING	The operation cannot be performed when the system is acquiring data.
TLA700_E_OUT_OF_MEMORY	There is not enough memory to perform the operation.
TLA700_E_SAVE_FILE_SIZE_LIMIT_EXCEEDED	The save operation cannot be performed because the size of the saved file will exceed the maximum limit supported by the file system.
TLA700_E_FAILED	The operation was unsuccessful.

Example

```
(Microsoft Visual Basic)

Dim App As Object
Dim Sys As Object
Dim LA As Object

Set App = CreateObject("TLA700.Application")
 'Get system.
Set Sys = App.GetSystem
 'Get module in slot 3.
Set LA = Sys.GetModuleBySlot(3)
 'Save module.
LA.SaveModule "C:\My Documents\a.tla","My module",1
```

Remarks

All file paths without machine qualifiers refer to drives mapped on the TLA instrument.

If the file already exists, it will be overwritten.

ILAModule::SetChannelGroup

Description

This method is used to set the channel list for a channel group, or to create a channel group and assign channels to it.

ODL Syntax

HRESULT SetChannelGroup([in] BSTR UserChannelGroupName,
[in] BSTR ChannelNameList)

UserChannelGroupName	This is the user defined name of the channel group whose channel list is to be set. If no channel group with this name exists, one will be created.
ChannelNameList	This is the list of channel names to be assigned to the channel group. For individual channels the syntax is the hardware pod name followed by the channel number enclosed in parentheses, (e.g. A0(1), A0(2), etc.). Groups of contiguous channels can be specified using a similar syntax, using a range of channel numbers within the parentheses, (e.g. A0(7-0), A1(3-5), etc.). When all of the channels in the hardware pod are to be specified, a shorthand notation is allowed using just an empty pair of parentheses, (e.g. A(), A1(), etc.). The syntax for clock and qualifier channels is the type identifier "CK or "Q" respectively, followed by a number, (e.g. CK0, CK1, Q0, Q1, etc.). Multiple channels or channel groups can be specified using a comma separated list. Embedded spaces are not allowed.

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
TLA700_E_INVALID_USER_CHANNEL_GROUP_NAME	Invalid “UserChannelGroupName” argument.
TLA700_E_INVALID_CHANNEL_NAME_LIST	Invalid “ChannelNameList” argument.
TLA700_E_OUT_OF_MEMORY	There is not enough memory to perform the operation.
TLA700_E_MODAL_DIALOG_OPEN	The operation was disallowed because a modal dialog is open.
TLA700_E_FAILED	The operation was unsuccessful.

Example

```
 (Microsoft Visual Basic)

Dim App As Object
Dim Sys As Object
Dim LA As Object

Set App = CreateObject("TLA700.Application")

'Get system.
Set Sys = App.GetSystem
      …
'Get module in slot 3.
Set LA = Sys.GetModuleBySlot(3)

'Create a channel group named "Address" and populate it
'with channels from pods A2, A1 and A0
LA.SetChannelGroup "Address", "A2(7-0),A1(7-0),A0(7-0)"

'Change the channel list for group "Address"
LA.SetChannelGroup "Address", "A3(7-0),A2(7-0),A1(7-0),A0(7-0)"
```

ILAModule::SetChannelName

Description

This method is used to set the user assigned name for a channel of this module.

ODL Syntax

```
HRESULT SetChannelName( [in] BSTR HWChannelName,
                        [in] BSTR UserChannelName )
```

Arguments

HWChannelName	This is the hardware name of the channel whose user name is to be set. For normal acquisition channels, the syntax is the hardware pod name followed by the channel number enclosed in parentheses, (e.g. A0(1), A0(2), etc.). The syntax for clock and qualifier channels is the type identifier "CK" or "Q" respectively, followed by a number, (e.g. CK0, CK1, Q0, Q1, etc.).
UserChannelName	This is the user name to be assigned to a specified channel. The user name must be unique among channels, and must not contain embedded spaces or commas.

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
TLA700_E_INVALID_HW_CHANNEL_NAME	Invalid “HWChannelName” argument.
TLA700_E_INVALID_USER_CHANNEL_NAME	Invalid “UserChannelName” argument.
TLA700_E_OUT_OF_MEMORY	There is not enough memory to perform the operation.
TLA700_E_MODAL_DIALOG_OPEN	Operation was disallowed because a modal dialog is open.
TLA700_E_FAILED	The operation was unsuccessful.

Example

(Microsoft Visual Basic)

```
Dim App As Object
Dim Sys As Object
Dim LA As Object

Set App = CreateObject("TLA700.Application")

'Get system.
Set Sys = App.GetSystem
      …
'Get module in slot 3.
Set LA = Sys.GetModuleBySlot(3)

'Set User channel name to "Reset"
'for probe C3, channel 4
LA.SetChannelName "C3(4)", "Reset"
```

Remarks

White space characters and comma "," are not allowed in channel names.

ILAModule::SetEventValue

Description

This method modifies the value(s) of a specified trigger event.

ODL Syntax

```
HRESULT  SetEventValue(  [in]  BSTR  EventID,
                         [in]  BSTR  EventValue  )
```

Arguments

`EventID` This string identifies the trigger event to be modified. Only recognizer values can be set. This is of the form: "<state>.<clause>.<event>"

For example: "1.2.3" specifies state 1, clause 2, event 3. State, clause and event numbers begin with 1.

The conditional storage clause is a special trigger state. It is specified using the following format "S.<event>"

For example: "S.3" specifies the third event of the storage clause.

`EventValue` This string specifies the event value(s). The format is dependent on the event type. It can consist of multiple subparts. Group and word values must be in hexadecimal. Timer values must be specified in picoseconds.

Group value: "aa" (for specifying single values) or "00,FF" (for specifying ranges)

Word value: "ffff,4444,aaaa" (when the setup defines three, 16-bit channel groups)

Counter value: "10"

Timer value: "4000"

Note: The TLA7Axx Series Logic Analyzer modules can have negative counter values. Therefore, counter values such as –200 are valid, but not for TLA600 Series Logic Analyzers or for TLA7Lx/Mx/Nx/Px/Qx Series Logic Analyzer modules. Passing a negative event value to a non TLA7Axx Series Logic Analyzer module counter, sets the target counter to zero; it does not cause an error.

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
TLA700_E_INVALID_EVENT_ID	Invalid “EventID” argument.
TLA700_E_INVALID_EVENT_VALUE	Invalid “EventValue” argument.
TLA700_E_INVALID_EVENT_TYPE	The specified event type is not supported.
TLA700_E_OUT_OF_TRIGGER_RESOURCES	The required trigger resources could not be allocated.
TLA700_E_OUT_OF_MEMORY	There is not enough memory to perform the operation.
TLA700_E_FAILED	The operation was unsuccessful.
TLA700_E_MODAL_DIALOG_OPEN	The operation was disallowed because a modal dialog is open.

Example

```
(Microsoft Visual Basic)
Dim App As Object
Dim Sys As Object
Dim LA As Object

Set App = CreateObject("TLA700.Application")
 'Get system.
Set Sys = App.GetSystem
 'Get module in slot 3.
Set LA = Sys.GetModuleBySlot(3)
 'Set event value.
LA.SetEventValue "1.2.3", "FF"
```

Remarks

Timer values must be in 4 ns increments, starting from 4 ns.

Event types, event operators, glitch settings, setup and hold group settings, and channel event values cannot be changed using this method; use ILAModule::LoadTrigger() to set these values.

Changed values do not take effect until the next acquisition. If an error value is returned, the event retains its original value.

ILAModule::SetTriggerPosition

Description

This method specifies a trigger position for the module.

ODL Syntax

```
HRESULT  SetTriggerPosition(  [in]  long  Position  )
```

Arguments

`Position` This is the trigger position specified as a percentage of total memory depth. For example: 50

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
TLA700_E_INVALID_TRIGGER_POSITION	Invalid "Position" argument.
TLA700_E_OUT_OF_MEMORY	There is not enough memory to perform the operation.
TLA700_E_FAILED	The operation was unsuccessful.
TLA700_E_MODAL_DIALOG_OPEN	The operation was disallowed because a modal dialog is open.

Example

```
(Microsoft Visual Basic)
Dim App As Object
Dim Sys As Object
Dim LA As Object

Set App = CreateObject("TLA700.Application")
 'Get system.
Set Sys = App.GetSystem
 'Get module in slot 3.
Set LA = Sys.GetModuleBySlot(3)
 'Set trigger position.
LA.SetTriggerPosition 50
```

Remarks

The new trigger position does not take effect until the next acquisition.

ISystem::DefineRangeSymbolOptions

Description

This method specifies the options to be used by ISystem::LoadSymbolFile when loading a range symbol file.

ODL Syntax

```
HRESULT DefineRangeSymbolOptions    ([in] long FileFormat,
                                     [in] long SymbolTypes,
                                     [in] long Reserved,
                                     [in] BSTR Bound1,
                                     [in] BSTR Bound2,
                                     [in] long OffsetType,
                                     [in] BSTR SymbolOffset )
```

Arguments (propget):

`FileFormat` – The format of the symbol file. This takes one of the following values:

HRESULT Return Codes

Return Code	Description
TLA700_AUTO_FORMAT (0)	Auto-detect the format
TLA700_TSF_FORMAT (1)	TSF
TLA700_IEEE695_FORMAT (2)	IEEE695
TLA700_OMF86_FORMAT (3)	OMF86
TLA700_OMF286_FORMAT (4)	OMF286
TLA700_OMF386_FORMAT (5)	OMF386
TLA700_COFF_FORMAT (6)	COFF
TLA700_ELF_FORMAT (7)	ELF
TLA700_OMF51_FORMAT (8)	OMF51
TLA700_OMF166_FORMAT (9)	OMF166

SymbolTypes - The types of symbols loaded. This takes one or any combination of the following values:

Value	Description
TLA700_FUNCTION_SYMBOLS (1)	Function symbols
TLA700_VARIABLE_SYMBOLS (2)	Variable symbols
TLA700_SOURCE_CODE_SYMBOLS (4)	Source Code symbols
TLA700_COLOR_SYMBOLS (8)	Color symbols
TLA700_ALL_SYMBOLS (0xFFFFFFFF)	All symbols

For example, to load function symbols and variable symbols, specify TLA700_FUNCTION_SYMBOLS + TLA700_VARIABLE_SYMBOLS or 3.

Reserved – `Reserved` for future use.

`Bound1, Bound2` – These arguments specify the range of symbols that will be loaded. Any strings representing hexadecimal values from "0" to "FFFFFFFF" may be specified.

`OffsetType` – The type of offset to be applied to the symbol values. The offset value can be one of those listed in the table below.

Value	Description
TLA700_DEFAULT_SYMBOL_OFFSET(0)	Default Offset (+0)
TLA700_CUSTOM_SYMBOL_OFFSET(1)	Custom Offset

TLA700_DEFAULT_SYMBOL_OFFSET applies only to TSF file formats and indicates that the offset should be read from the TSF file header.

For TLA700_CUSTOM_SYMBOL_OFFSET, the actual value of a custom offset is specified in the SymbolOffset argument.

`SymbolOffset` – The value of a custom offset. This is specified as a string and takes the form "+N" or "N" or "-N" where N is the value of the offset in hexadecimal and can be from 0 to 0xFFFFFFFF. + indicates that the offset is to be added and – indicates that the offset is to be subtracted. If a sign is not specified, + is assumed.

This argument is ignored if OffsetType is equal to TLA700_DEFAULT_SYMBOL_OFFSET.

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
TLA700_E_INVALID_FILE_FORMAT	Invalid “FileFormat” argument.
TLA700_E_INVALID_SYMBOL_TYPES	Invalid “SymbolTypes” argument.
TLA700_E_INVALID_BOUND1	Invalid “Bound1” argument.
TLA700_E_INVALID_BOUND2	Invalid “Bound2” argument.
TLA700_E_INVALID_OFFSET_TYPE	Invalid “OffsetType” argument.
TLA700_E_INVALID_SYMBOL_OFFSET	Invalid “SymbolOffset” argument.
TLA700_E_OUT_OF_MEMORY	There is not enough memory to perform the operation.
TLA700_E_FAILED	The operation was unsuccessful.
TLA700_E_MODAL_DIALOG_OPEN	The operation was disallowed because a modal dialog is open.

Example

```
 (Microsoft Visual Basic)

 Dim App As Object
 Dim Sys As Object

 Set App = CreateObject("TLA700.Application")
  'Get system.
 Set Sys = App.GetSystem
  'Set range symbol options
 `      - TSF file format
 `      - Function and variable symbols
 `      - Bound1 = 0 and Bound2 = FFFF
 `      - Custom offset, Add 0F00
 Sys.DefineRangeSymbolOptions 1, 3, 0, "0", "FFFF", 1,
    "0F00"
Sys.LoadSymbolFile "C:\My Symbol File.tsf"
```

Remarks

These options apply only to range symbol files and do not apply to pattern symbol files.

If these options are not explicitly set, the following default range symbol options are used by ISystem::LoadSymbolFile:

FileFormat:	TLA700_AUTO_FORMAT
SymbolTypes:	TLA700_ALL_SYMBOLS
Reserved:	0
Bound1:	0
Bound2:	FFFFFFFF
OffsetType:	Default Offset
SymbolOffset:	0

ISystem::ExternalSignalIn

Description

This property is used to specify which Internal Signal, if any, should be connected to External Signal In, or to get the current setting for External Signal In.

ODL Syntax

```
[
   propget
]
HRESULT ExternalSignalIn( [out, retval] long* pInternalSignal )

[
   propput
]
HRESULT ExternalSignalIn( [in] long InternalSignal )
```

Arguments (propget):

`pInternalSignal - The Internal Signal currently connected to External Signal In.`

Arguments (propput):

`InternalSignal - The Internal Signal to be connected to External Signal In.`

Value	Description
TLA700_INTERNAL_SIGNAL_ NONE (0)	None (No internal signal)
TLA700_INTERNAL_SIGNAL_ 1 (1)	Signal 1 – High Speed
TLA700_INTERNAL_SIGNAL_ 2 (2)	Signal 2- High Speed
TLA700_INTERNAL_SIGNAL_ 3 (3)	Signal 3
TLA700_INTERNAL_SIGNAL_ 4 (4)	Signal 4

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
TLA700_E_INVALID_INTERNAL_SIGNAL	Invalid “InternalSignal” argument.
TLA700_E_OUT_OF_MEMORY	There is not enough memory to perform the operation.
TLA700_E_MODAL_DIALOG_OPEN	The operation was disallowed because a modal dialog is open.
TLA700_E_FAILED	The operation was unsuccessful.

Example

```
(Microsoft Visual Basic)
```

```
Dim App As Object
Dim Sys As Object
Dim InternalSignal As Long

Set App = CreateObject("TLA700.Application")

‘Get system.
Set Sys = App.GetSystem
        …
‘Get External Signal In setting.
InternalSignal = Sys.ExternalSignalIn

‘Set External Signal In to Signal 3
Sys.ExternalSignalIn = TLA700_INTERNAL_SIGNAL_3
```

ISystem::ExternalSignalOut

Description

This property is used to specify which Internal Signal, if any, should be connected to External Signal Out, or to get the current setting for External Signal Out.

ODL Syntax

```
[
   propget
]
HRESULT ExternalSignalOut( [out, retval] long* pInternalSignal )

[
   propput
]
HRESULT ExternalSignalOut( [in] long InternalSignal )
```

Arguments (propget):

`pInternalSignal - The Internal Signal currently connected to External Signal Out.`

Arguments (propput):

`InternalSignal - The Internal Signal connected to External Signal Out.`

Value	Description
TLA700_INTERNAL_SIGNAL_ NONE (0)	None (No internal signal)
TLA700_INTERNAL_SIGNAL_ 1 (1)	Signal 1 – High Speed
TLA700_INTERNAL_SIGNAL_ 2 (2)	Signal 2- High Speed
TLA700_INTERNAL_SIGNAL_ 3 (3)	Signal 3
TLA700_INTERNAL_SIGNAL_ 4 (4)	Signal 4
TLA700_INTERNAL_SIGNAL_10 MHz_CLOCK (5)	10 MHz Clock

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
TLA700_E_INVALID_INTERNAL_SIGNAL	Invalid “InternalSignal” argument.
TLA700_E_OUT_OF_MEMORY	There is not enough memory to perform the operation.
TLA700_E_MODAL_DIALOG_OPEN	The operation was disallowed because a modal dialog is open.
TLA700_E_FAILED	The operation was unsuccessful.

Example

(Microsoft Visual Basic)

```
Dim App As Object
Dim Sys As Object
Dim InternalSignal As Long

Set App = CreateObject("TLA700.Application")

‘Get system.
Set Sys = App.GetSystem
      …
‘Get External Signal Out setting.
InternalSignal = Sys.ExternalSignalOut

‘Set External Signal Out to Signal 1
Sys.ExternalSignalOut = TLA700_INTERNAL_SIGNAL_1
```

ISystem::ExternalSignalOutLowTrue

Description

This property is used to get or set logical polarity for External Signal Out. When True, the External Signal Out is low when asserted.

ODL Syntax

```
[
   propget
]
HRESULT ExternalSignalOutLowTrue( [out, retval] VARIANT_BOOL* pLowTrue
)

[
   propput
]
HRESULT ExternalSignalOutLowTrue ( [in] VARIANT_BOOL LowTrue)
```

Arguments (propget):

`PLowTrue`	The logical polarity of External Signal Out. When True, then the External Signal Out is low when asserted.

Arguments (propput):

`LowTrue`	The logical polarity of External Signal Out. When True, then the External Signal Out is low when asserted.

Value	Description
TRUE	External Signal Out Polarity is Low True
FALSE	External Signal Out Polarity is High True

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
TLA700_E_OUT_OF_MEMORY	There is not enough memory to perform the operation.
TLA700_E_MODAL_DIALOG_OPEN	The operation was disallowed because a modal dialog is open.
TLA700_E_FAILED	The operation was unsuccessful.

Example

```
(Microsoft Visual Basic)

Dim App As Object
Dim Sys As Object
Dim LowTrue As Boolean

Set App = CreateObject("TLA700.Application")

'Get system.
Set Sys = App.GetSystem
        ...
'Get External Signal Out Polarity setting.
LowTrue = Sys.ExternalSignalOutLowTrue

'Set External Signal Out Polarity to Low True
Sys.ExternalSignalOutLowTrue = True
```

ISystem::GetDiagCalStatus

Description

This method returns the power-on diagnostics and calibration status. The results do not include information for external oscilloscope modules.

ODL Syntax

```
HRESULT GetDiagCalStatus( [out] BSTR* pDiagCalStatus )
```

Arguments

`pDiagCalStatus` The status of power-on diagnostics and calibration.This is of the form: "<diagnostics status>,<calibration status>"
For example: "Pass,Calibrated"
The power-on diagnostics status can take one of the following values: "Running", "Pass", "Fail", "Mixed", or "Unknown"
The calibration status can take one of the following values: "Calibrated", "Unknown", "Failed", "Running", or "No Calibration Required"

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
TLA700_E_OUT_OF_MEMORY	There is not enough memory to perform the operation.
TLA700_E_FAILED	The operation was unsuccessful.

Example

```
(Microsoft Visual Basic)

Dim App As Object
Dim Sys As Object
Dim Status As String

Set App = CreateObject("TLA700.Application")
```

```
‘Get system.
Set Sys = App.GetSystem

‘Get diag/cal status.
Status = Sys.GetDiagCalStatus
```

Remarks

The TLA server allocates space for the returned string. The client is responsible for freeing the space when it is no longer in use.

It is recommended that the client program examine the power-on diagnostics and calibration status and only use the TLA instrument if the value is "Pass, Calibrated". The TLA instrument will not function properly if it fails power-on diagnostics or is not calibrated

ISystem::GetFirstModuleSlot

Description

This method returns the number of the first slot in the mainframe.

ODL Syntax

```
HRESULT GetFirstModuleSlot( [out, retval] long* pSlot )
```

Arguments

`pSlot` The number of the first slot in the mainframe.

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
TLA700_E_OUT_OF_MEMORY	There is not enough memory to perform the operation.
TLA700_E_FAILED	The operation was unsuccessful.

Example

```
(Microsoft Visual Basic)

Dim App As Object
Dim Sys As Object
Dim FirstModuleSlot As Long

Set App = CreateObject("TLA700.Application")
 ‘Get system.
Set Sys = App.GetSystem
 ‘Get first slot.
FirstModuleSlot = Sys.GetFirstModuleSlot
```

Remarks

The value returned is the slot number of the first slot.

For the TLA714 and TLA704 Color Portable Mainframe and the TLA600 series logic analyzer, the slot number is 1.

For the TLA711 and TLA720 Color Benchtop Mainframe the slot number is 0.

Refer to Slot Numbers for Expansion Mainframes for more information on how to specify slot numbers with expansion mainframes.

ISystem::GetModuleByName

Description

This method returns the interface pointer for the logical module with the specified name. The module name should be as specified in the TLA System window.

ODL Syntax

```
HRESULT  GetModuleByName(  [in]  BSTR  ModuleName,
                                                    [out,  retval]
IDispatch**  ppDispatch  )
```

Arguments

`ModuleName` The name of the required module. This is the module name that is shown in the TLA System window.

`ppDispatch` The interface pointer for the module with the specified name.

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
TLA700_E_INVALID_MODULE_NAME	Invalid “ModuleName” argument.
TLA700_E_OUT_OF_MEMORY	There is not enough memory to perform the operation.
TLA700_E_FAILED	The operation was unsuccessful.

Example

```
(Microsoft Visual Basic)

Dim App As Object
Dim Sys As Object
Dim LA As Object

Set App = CreateObject("TLA700.Application")
 ‘Get system.
Set Sys = App.GetSystem
 ‘Get module.
Set LA = Sys.GetModuleByName("LA 1")
```

Remarks

Module references obtained using this method are invalidated by operations such as ISystem::LoadSystem that affect the logical modules in the system. Remember to release any module references before performing such operations.

ISystem::GetModuleBySlot

Description

This method returns the interface pointer for the logical module in the specified slot.

ODL Syntax

```
HRESULT  GetModuleBySlot(  [in]  long  Slot,
                           [out,  retval]  IDispatch**  ppDispatch  )
```

Arguments

`Slot` The slot number. This can correspond to any of the slots occupied by the logical module.

`ppDispatch` The interface pointer for the module in the specified slot.

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
TLA700_E_EMPTY_SLOT	The specified slot is empty.
TLA700_E_UNKNOWN_MODULE	The module in the specified slot is not recognized.
TLA700_E_INVALID_SLOT_NUMBER	Invalid “Slot” argument.
TLA700_E_OUT_OF_MEMORY	There is not enough memory to perform the operation.
TLA700_E_FAILED	The operation was unsuccessful.

Example

```
(Microsoft Visual Basic)

Dim App As Object
Dim Sys As Object
Dim LA As Object

Set App = CreateObject("TLA700.Application")
 'Get system.
Set Sys = App.GetSystem
 'Get module in slot 3.
Set LA = Sys.GetModuleBySlot(3)
```

Remarks

Refer to Slot Numbers for Expansion Mainframes for more information on how to specify slot numbers with expansion mainframes.

Module references obtained by this method are invalidated by operations such as ISystem::LoadSystem that affect the logical modules in the system. Remember to release any module references before performing such operations.

This method can not be used with external oscilloscopes.

ISystem::GetModuleNames

Description

This method retrieves the names of all logical modules in the system, including external oscilloscope modules.

ODL Syntax

```
HRESULT GetModuleNames( [out, retval] VARIANT* pModuleNames )
```

Arguments

`pModuleNames` – The module names.

Module names are returned as a VARIANT. The variant is of type VT_ARRAY and points to a SAFEARRAY. The SAFEARRAY has dimension 1 and the elements are of type VT_BSTR. The number of modules is equal to the number of elements in the SAFEARRAY.

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
TLA700_E_OUT_OF_MEMORY	There is not enough memory to perform the operation.
TLA700_E_FAILED	The operation was unsuccessful.

Example

```
(Microsoft Visual Basic)

Dim App As Object
Dim Sys As Object
Dim M As Variant
Dim Modules As Variant

Set App = CreateObject("TLA700.Application")
 'Get system.
Set Sys = App.GetSystem
 'Get module names.
Modules = Sys.GetModuleNames
 `Access module names.
For Each M In Modules
       'Use module name in M.
Next M
```

Remarks

If there are no modules, the SAFEARRAY returned will be empty.

ISystem::GetModulePropertiesBySlot

Description

This method returns the properties of the physical module in the specified slot.

ODL Syntax

```
HRESULT GetModulePropertiesBySlot( [in] long Slot,
                    [out, retval] BSTR* pModuleProperties )
```

Arguments

`Slot` The slot number.

`pModuleProperties` The properties of the physical module in the specified slot. This is of the format shown below. Fields are included as they apply.
"<manufacturer>,<model>,<firmware version>,
<diagnostics status>,<calibration status>,
<speed>,<memory depth>"

For example:

LA	"Tektronix,TLA7N4,2.0.1,Pass,Calibrated,100 MHz, 32K"
DSO	Tektronix,TLA7E2,2.0.2,Fail,Calibrated,2500 MS/s, 15000"
Controller	"Tektronix,TLA711,2.0.033,Pass"
Unknown module	"0ffd,5d07"

Expansion Interface module: "Tektronix TLA7XM,Pass"
Refer to ISystem::GetDiagCalStatus for possible values for diagnostics status and calibration status.

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
TLA700_E_EMPTY_SLOT	The specified slot is empty.
TLA700_E_INVALID_SLOT_NUMBER	Invalid “Slot” argument.
TLA700_E_OUT_OF_MEMORY	There is not enough memory to perform the operation.
TLA700_E_FAILED	The operation was unsuccessful.

Example

```
(Microsoft Visual Basic)

Dim App As Object
Dim Sys As Object
Dim ModDesc As String

Set App = CreateObject("TLA700.Application")
 'Get system.
Set Sys = App.GetSystem
 'Get description of module in slot 3.
ModDesc = Sys.GetModulePropertiesBySlot(3)
```

Remarks

For modules that occupy more than one slot, the same string is returned for each of the slots.

Refer to Slot Numbers for Expansion Mainframes for more information on how to specify slot numbers with expansion mainframes.

The TLA server allocates space for the returned string. The client is responsible for freeing the space when it is no longer in use.

Do not use a module that does not pass power-on diagnostics or is not calibrated.

This method cannot be used with external oscilloscopes.

ISystem::GetModuleSlotByName

Description

This method returns the slot number for the (logical) module with specified name. The slot number returned is the lowest numbered slot occupied by the module.

ODL Syntax

```
HRESULT GetModuleSlotByName( [in] BSTR ModuleName,
                             [out, retval] long* pSlot )
```

Arguments

`ModuleName` – The user name of the required module. This is the name that you will see in the System Window.

`pSlot` – The slot number occupied by the module with the specified name.

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
TLA700_E_INVALID_SLOT_NUMBER	Invalid "Slot" argument.
TLA700_E_OUT_OF_MEMORY	There is not enough memory to perform the operation.
TLA700_E_FAILED	The operation was unsuccessful.

Example

`(Microsoft Visual Basic)`

```
Dim App As Object
Dim Sys As Object
Dim ModuleSlot As Long

Set App = CreateObject("TLA700.Application")

'Get system.
Set Sys = App.GetSystem
        ...
'Get module slot number.
ModuleSlot = Sys.GetModuleSlotByName "LA 1"
```

Remarks

The slot value returned is the lowest numbered slot occupied by the physical module. If the named module is part of a merged set of modules, the slot number returned is that corresponding to the master module.

ISystem::GetModuleTypeBySlot

Description

This method returns the type of the physical module in the specified slot.

ODL Syntax

```
HRESULT  GetModuleTypeBySlot(  [in]  long  Slot,
                               [out,  retval]  long*  pModuleType  )
```

Arguments

`Slot`	The slot number.
`pModuleType`	The type of the physical module in the specified slot. This can be one of the following values:

HRESULT Return Codes

Return Value	Description
TLA700_LA_MODULE (0)	LA module
TLA700_DSO_MODULE (1)	DSO module
TLA700_CONTROLLER_MODULE (2)	Controller module
TLA700_UNKNOWN_MODULE (3)	Unknown module
TLA700_EMPTY_SLOT (4)	Empty slot
TLA700_EXPANSION_INTERFACE_MODULE(5)	Expansion Interface module

Return Code	Description
S_OK	The operation succeeded.
TLA700_E_INVALID_SLOT_NUMBER	Invalid “Slot” argument.
TLA700_E_OUT_OF_MEMORY	There is not enough memory to perform the operation.
TLA700_E_FAILED	The operation was unsuccessful.

Example

```
(Microsoft Visual Basic)

Dim App As Object
Dim Sys As Object
Dim ModType As Long

Set App = CreateObject("TLA700.Application")
 ‘Get system.
Set Sys = App.GetSystem
 ‘Get type of module in slot 3.
ModType = Sys.GetModuleTypeBySlot(3)
```

Remarks

For instrument modules that occupy more than one slot, the same module type is returned for each of its slots.

Refer to Slot Numbers for Expansion Mainframes for more information on how to specify slot numbers with expansion mainframes.

This method can not be used with external oscilloscopes.

ISystem::GetNumModuleSlots

Description

This method returns the total number of slots in the TLA mainframe.

ODL Syntax

```
HRESULT GetNumModuleSlots( [out, retval] long* pNumSlots )
```

Arguments

`pNumSlots` The number of slots in the mainframe.

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
TLA700_E_OUT_OF_MEMORY	There is not enough memory to perform the operation.
TLA700_E_FAILED	The operation was unsuccessful.

Example

```
(Microsoft Visual Basic)

Dim App As Object
Dim Sys As Object
Dim NumModuleSlots As Long

Set App = CreateObject("TLA700.Application")
 'Get system.
Set Sys = App.GetSystem
 'Get number of slots.
NumModuleSlots = Sys.GetNumModuleSlots
```

Remarks

For the Portable Mainframe and the TLA600 series logic analyzer, the value returned is 4.

For the Benchtop Mainframe the value returned is 13.

Refer to Slot Numbers for Expansion Mainframes for more information on how to specify slot numbers with expansion mainframes.

The number of module slots does not include external oscilloscopes.

ISystem::GetRepetitiveStopReason

Description

This method is used to determine why the last repetitive acquisition stopped.

ODL Syntax

```
HRESULT GetRepetitiveStopReason( [out, retval] long* pStopReason )
```

Arguments

`pRepetitiveStopReason` – Value indicating why the last repetitive acquisition stopped. Refer to the following table:

Value	Description
TLA700_REPEAT_STOP_UNKNOWN (0)	Repetitive acquisition stopped for an unknown reason.
TLA700_REPEAT_STOP_USER_REQUEST (1)	Repetitive acquisition was halted by a user STOP request, either through the user interface or via TPI.
TLA700_REPEAT_STOP_COUNT (2)	Repetitive acquisition was halted because the repeat count was reached.
TLA700_REPEAT_STOP_COMPARE (3)	Repetitive acquisition was halted because the compare test succeeded.
TLA700_REPEAT_STOP_SAVE_FAILED (4)	Repetitive acquisition was halted because an attempt to save data failed.

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
TLA700_E_OUT_OF_RUNNING	The operation cannot be performed when the system is still in Run mode.
TLA700_E_OUT_OF_MEMORY	There is not enough memory to perform the operation.
TLA700_E_FAILED	The operation was unsuccessful.

Example

(Microsoft Visual Basic)

```
Dim App As Object
Dim Sys As Object
Dim StopReason As Long

Set App = CreateObject("TLA700.Application")

'Get system.
Set Sys = App.GetSystem
      …

'Get reason why repetitive acquisition ended.
StopReason = Sys.GetRepetitiveStopReason
```

ISystem::GetRunStatus

Description

This method returns the current run status of the TLA server.

ODL Syntax

```
HRESULT    GetRunStatus(  [out,  retval]  long*  pRunStatus  )
```

Arguments

`pRunStatus` The current run status. This can be one of the values in the following table:

Return Value	Description
TLA700_ACQ_RUNNING (0)	An acquisition has been started and is currently running.
TLA700_ACQ_IDLE (1)	No acquisition is currently running. Any acquisitions that were previously running have completed or have been stopped.

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
TLA700_E_OUT_OF_MEMORY	There is not enough memory to perform the operation.
TLA700_E_FAILED	The operation was unsuccessful.

Example

```
(Microsoft Visual Basic)

Dim App As Object
Dim Sys As Object
Dim RunStatus As Long

Set App = CreateObject("TLA700.Application")
 'Get system.
Set Sys = App.GetSystem
 'Start acquisition and wait until it is complete.
Sys.Run
Do
    RunStatus = Sys.GetRunStatus
Loop While (RunStatus = 0)
```

ISystem::GetSWVersion

Description

This method returns the software version of the server.

ODL Syntax

```
HRESULT GetSWVersion( [out, retval] BSTR* pVersion )
```

Arguments

`pVersion` The software version of the server. This is in the form: "<major no.>.<minor no.>.<build no.>". For example, `"2.0.112"`.

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
TLA700_E_OUT_OF_MEMORY	There is not enough memory to perform the operation.
TLA700_E_FAILED	The operation was unsuccessful.

Example

```
(Microsoft Visual Basic)

Dim App As Object
Dim Sys As Object
Dim SWVersion As String

Set App = CreateObject("TLA700.Application")
 'Get system.
Set Sys = App.GetSystem
 'Get software version.
SWVersion = Sys.GetSWVersion
```

Remarks

The TLA server allocates space for the returned string. The client is responsible for freeing the space when it is no longer in use.

ISystem::LoadSymbolFile

Description

This method loads a symbol file into the system.

ODL Syntax

```
HRESULT LoadSymbolFile( [in] BSTR SymbolFilePath )
```

Arguments

`SymbolFilePath` - The full path to the required symbol file.
Eg: "C:\My Documents\My Symbol File.tsf"

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
TLA700_E_LOAD_INVALID_FILE	An error occurred opening the file for reading.
TLA700_E_OUT_OF_MEMORY	There is not enough memory to perform the operation.
TLA700_E_FAILED	The operation was unsuccessful.
TLA700_E_MODAL_DIALOG_OPEN	The operation was disallowed because a modal dialog is open.

Example

```
 (Microsoft Visual Basic)

 Dim App As Object
 Dim Sys As Object

 Set App = CreateObject("TLA700.Application")

 'Get system.
 Set Sys = App.GetSystem
  'Set range symbol options
  `      - TSF file format
  `      - Function and variable symbols
  `      - Bound1 = 0 and Bound2 = FFFF
  `      - Custom offset, Add 0F00
 Sys.DefineRangeSymbolOptions 1, 3, 0, "0", "FFFF", 1,
    "0F00"
Sys.LoadSymbolFile "C:\My Symbol File.tsf"
```

Remarks

All file paths without machine qualifiers refer to drives mapped on the TLA700.

The options to be used when loading a range symbol file may be set by invoking ISystem::DefineRangeSymbolOptions before calling this method.

If these options are not explicitly set, the following values are used by default:

FileFormat:	TLA700_AUTO_FORMAT
SymbolTypes:	TLA700_ALL_SYMBOLS
Reserved:	0
Bound1:	0
Bound2:	FFFFFFFF
OffsetType:	Default Offset
SymbolOffset:	0

ISystem::LoadSystem

Description

This method loads the TLA server with the specified TLA system file.

ODL Syntax:

```
HRESULT  LoadSystem(  [in]  BSTR  SystemPath  )
```

Arguments

`SystemPath` The full path to the required TLA system file. For example: **"C:\My Documents\My System.tla"**

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
TLA700_E_LOAD_INVALID_FILE	An error occurred opening the file for reading.
TLA700_E_LOAD_MISMATCH	The system configuration in the file does not match the current hardware configuration.
TLA700_E_LOAD_MODULE_ERROR	An error occurred loading a module from the file.
TLA700_E_LOAD_DATA_ERROR	An error occurred loading data from the file.
TLA700_E_LOAD_DATA_WINDOW_ERROR	An error occurred loading a data window from the file.
TLA700_E_LOAD_ERROR	An error occurred retrieving information from the file during the load operation.
TLA700_E_SYSTEM_RUNNING	The operation cannot be performed when the system is acquiring data.
TLA700_E_OUT_OF_MEMORY	There is not enough memory to perform the operation.
TLA700_E_FAILED	The operation was unsuccessful.
TLA700_E_MODAL_DIALOG_OPEN	The operation was disallowed because a modal box is open.

Example

```
(Microsoft Visual Basic)

Dim App As Object
Dim Sys As Object

Set App = CreateObject("TLA700.Application")
 'Get system.
Set Sys = App.GetSystem
 'Load system.
Sys.LoadSystem "C:\My Documents\System1.tla"
```

Remarks

All file paths without machine qualifiers refer to drives mapped on the TLA instrument.

After invoking LoadSystem(), all module references previously obtained using calls to GetModuleBySlot() and GetModuleByName() are invalid.

Client Applications must note that the focus may be transferred to the TLA application window as a result of invoking LoadSystem().

Under some circumstances, when this operation is performed under the TLA Application Graphical User Interface, the user will be prompted to approve changes to some trigger actions. The purpose of the proposed changes (if any) is to adjust individual module trigger programs for compatibility with the System Trigger specification. When this operation is performed using the programmatic (TPI), no user prompt is possible and restored trigger programs will not be altered.

ISystem::Repetitive

Description

This property allows the client to specify the repetitive acquisition setting of the system.

ODL Syntax

```
[
propget
]
HRESULT Repetitive( [out, retval] VARIANT_BOOL* pRepetitive )

[
propput
]
HRESULT Repetitive( [in] VARIANT_BOOL Repetitive )
```

Arguments (propget):

`pRepetitive` - The repetitive acquisition setting of the system

Arguments (propput):

`Repetitive` - The repetitive acquisition setting of the system

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
TLA700_E_OUT_OF_MEMORY	There is not enough memory to perform the operation.
TLA700_E_FAILED	The operation was unsuccessful.
TLA700_E_MODAL_DIALOG_OPEN	The operation was disallowed because the dialog box is open.

Example

```
(Microsoft Visual Basic)

Dim App As Object
Dim Sys As Object
Dim Repetitive As Boolean

Set App = CreateObject("TLA700.Application")
 'Get system.
Set Sys = App.GetSystem
 'Get repetitive setting.
Repetitive = Sys.Repetitive
 'Set repetitive acquisition setting.
Sys.Repetitive = True
```

ISystem::Run

Description

This method starts a data acquisition operation. This method is equivalent to pressing the Run button in the TLA graphical user interface.

ODL Syntax

```
HRESULT  Run()
```

Arguments

None

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
TLA700_E_NO_ENABLED_MODULES	There are no enabled modules in the current system.
TLA700_E_SYSTEM_RUNNING	The operation cannot be performed when the system is acquiring data.
TLA700_E_OUT_OF_MEMORY	There is not enough memory to perform the operation.
TLA700_E_FAILED	The operation was unsuccessful.
TLA700_E_MODAL_DIALOG_OPEN	The operation was disallowed because the dialog box is open.

Example

```
(Microsoft Visual Basic)

Dim App As Object
Dim Sys As Object
Dim RunStatus As Long

Set App = CreateObject("TLA700.Application")
 'Get system.
Set Sys = App.GetSystem
 'Start acquisition and wait until it is complete.
Sys.Run
Do
    RunStatus = Sys.GetRunStatus
Loop While (RunStatus = 0)
```

Remarks

This method starts a data acquisition operation but does not wait for it to complete before returning. Note that many operations, such as ISystem::LoadSystem and ILAModule::GetData , cannot be performed while the TLA is running. After calling the ISystem::Run () method, the ISystem::GetRunStatus() method can be used to find out the current run status of the TLA instrument.

ISystem::RunCount

Description

This property allows the client to retrieve the total number of acquisitions that were run in the system during the current session of the application.

ODL Syntax

```
[
propget
]
HRESULT RunCount( [out, retval] long* pRunCount)
```

Arguments (propget):

`pRunCount` - The total number of acquisitions run in the system

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
TLA700_E_OUT_OF_MEMORY	There is not enough memory to perform the operation.
TLA700_E_FAILED	The operation was unsuccessful.

Example

```
(Microsoft Visual Basic)

Dim App As Object
Dim Sys As Object
Dim RunCount As Long

Set App = CreateObject("TLA700.Application")
 'Get system.
Set Sys = App.GetSystem
 'Get run count.
RunCount = Sys.RunCount
```

Remarks

This method returns the total number of acquisitions that have occurred so far during the current session of the application. The acquisitions in a repetitive cycle are counted together as one.

ISystem::SaveSystem

Description

This method saves the TLA system to a file.

ODL Syntax

```
HRESULT  SaveSystem(  [in]  BSTR  SystemPath,
                      [in]  BSTR  UserComment,
                      [in]  long  SaveData  )
```

Arguments

`SystemPath` The full path to the destination system file.
For example: **"C:\My Documents\My System.tla"**

`UserComment` The user comment to be saved in the file.

`SaveData` This flag takes one of the following values:

Value	Description
TLA700_SAVE_NO_DATA (0)	Do not save acquisition data in file.
TLA700_SAVE_DATA (1)	Save acquisition data in file.
TLA700_SAVE_UNSUPPRESSED_DATA (2)	Save only unsuppressed acquisition data in file.

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
TLA700_E_INVALID_SAVE_DATA	Invalid "SaveData" argument.
TLA700_E_ SAVE_INVALID_FILE	An error occurred opening this file for writing.
TLA700_E_SAVE_OUT_OF_SPACE	There is not enough disk space to perform this operation.
TLA700_E_SAVE_ERROR	An error occurred during the save operation.
TLA700_E_SYSTEM_RUNNING	The operation cannot be performed when the system is acquiring data.
TLA700_E_OUT_OF_MEMORY	There is not enough memory to perform the operation.
TLA700_E_SAVE_FILE_SIZE_LIMIT_EXCEEDED	The save operation cannot be performed because the size of the saved file will exceed the maximum limit supported by the file system.
TLA700_E_FAILED	The operation was unsuccessful.

Example

```
(Microsoft Visual Basic)

Dim App As Object
Dim Sys As Object

Set App = CreateObject("TLA700.Application")
 'Get system.
Set Sys = App.GetSystem
 'Save system with data.
Sys.SaveSystem "C:\My Documents\a.tla", "My system", 1
```

Remarks

All file paths without machine qualifiers refer to drives mapped on the TLA instrument.

If the file already exists, it will be overwritten. The resulting file can be loaded later by using ISystem::LoadSystem.

ISystem::Stop

Description

This method stops a data acquisition operation. This method is equivalent to pressing the Stop button in the TLA graphical user interface.

ODL Syntax

```
HRESULT  Stop()
```

Arguments

None

HRESULT Return Codes

Return Code	Description
S_OK	The operation succeeded.
TLA700_E_SYSTEM_NOT_RUNNING	The system is not running.
TLA700_E_OUT_OF_MEMORY	There is not enough memory to perform the operation.
TLA700_E_FAILED	The operation was unsuccessful.
TLA700_E_MODAL_DIALOG_OPEN	The operation was disallowed because a modal dialog is open.

Example

```
(Microsoft Visual Basic)

Dim App As Object
Dim Sys As Object
Dim RunStatus As Long

Set App = CreateObject("TLA700.Application")

‘Get system.
Set Sys = App.GetSystem
 ‘Start acquisition.
Sys.Run
 ‘Stop acquisition.
Sys.Stop
 ‘Wait until acquisition stops.
Do
    RunStatus = Sys.GetRunStatus
Loop While (RunStatus = 0)
```

Remarks

This method issues a request to stop the system but does not wait for the stop operation to complete. After calling this method, the method ISystem::GetRunStatus() can be used to find out the current run status of the TLA instrument.

Data Formats

Data Formats

Binary data formats for LA modules and the TLA600 series logic analyzer

General characteristics

Delimiters are not used between samples in the binary data formats for LA modules. Each sample is a fixed size.

Note: You must use ILAModule::DefineDataFormat to specify a binary data format. You can then use ILAModule::GetData or ILAModule::ExportData to get the data in the specified format.

Violation data

Data formats for the violation data set are identical to those for the main data set. For the violation data set, a 1 in a bit position indicates that a either a glitch or a setup hold violation occurred on the corresponding channel.

Time stamps

A time stamp value represents the time at which the sample was stored relative to the start of the acquisition. It is important to note that the start of the acquisition is not the same as the first sample of data. The start of the acquisition is the moment at which the system is enabled to begin acquiring data. It takes some finite amount of time after the start of the acquisition before a module can actually store a sample of data (especially when using an external clock).

When acquisition data is returned in a raw format, time stamp values are returned in units of ticks where each tick represents a fixed amount of time. To obtain the time stamp values in picoseconds, it must be adjusted as follows:

*Time stamp value in ps = (Time stamp value in ticks *Time stamp Multiplier)+Expansion mainframe Offset +User-defined time alignment in ps*

The Time stamp Multiplier is obtained by calling ILAModule::GetTimestampMultiplier(). Expansion Mainframe Offset is 36000 ps if the module is in an expansion mainframe or 0 ps if it is not. User-defined time alignment is any time alignment defined for the module by the user.

When acquisition data is returned in a grouped format, time stamp values are returned in picoseconds.

RawWithTimestamp. This format is available only for the main data set and violation data set. It is not available for the MagniVu data set, since the MagniVu data set does not contain time stamp data. The format is identical for both the main data set and the violation data set.

RawWithoutTimestamp. This format does not contain time stamp bytes and is available for the main data set, violation data set and MagniVu data set. The format is identical for both the main data set and the violation data set. The MagniVu data format is narrower because of the absence of status bytes.

AllGroupsWithTimestamp. In this data format, data for all channel groups is returned in the order in which the groups are defined in the LA Setup window. In each sample, time stamp data is returned after the data for the groups. Channel groups are zero padded to the nearest byte boundary.

See AllGroupsWithTimestamp for an example of this binary data format.

AllGroupsWithoutTimestamp. In this data format, data for all channel groups is returned in the order in which the groups are defined in the LA Setup window. Time stamp data is not returned. Channel groups are zero padded to the nearest byte boundary.

See AllGroupsWithoutTimestamp for an example of this binary data format.

GroupList. In this data format, data for specified channel groups is returned in the order in which the groups were specified. Channel groups are zero padded to the nearest byte boundary.

See GroupList for an example of this binary data format.

RawWithTimestamp binary data format for TLA7N1 LA modules and the TLA6X1 logic analyzer

A main data sample for a stand-alone module consists of 14 bytes, starting with C3(7-0) and ending with Timestamp(7-0). The byte labeled "Reserved 15-8" contains the gap bit. This bit is set TRUE when there is a qualification gap between the current sample and the previous sample.

RawWithTimestamp Binary Data			
C3 7 6 5 4 3 2 1 0	C2 7 6 5 4 3 2 1 0	A3 7 6 5 4 3 2 1 0	A2 7 6 5 4 3 2 1 0
Clock Quals (See ordering)	Reserved 15–8	Reserved 7–0	Timestamp 5 pad + 50–48
Timestamp 47–40	Timestamp 39–32	Timestamp 31–24	Timestamp 23–16
Timestamp 15–8	Timestamp 7–0		

Clock/Quals Byte							
7 Pad	6 Pad	5 Pad	4 Pad	3 CK3	2 Pad	1 Pad	0 CK0

Reserved 15-8 Byte							
7 Resv'd	6 Resv'd	5 Resv'd	4 Resv'd	3 Gap	2 Resv'd	1 Resv'd	0 Resv'd

RawWithTimestamp binary data format for TLA7N2/P2/Q2 LA modules and the TLA6X2 logic analyzer

A main data sample for a stand-alone module consists of 18 bytes, starting with A1(7-0) and ending with time stamp(7-0).). The byte labeled "Reserved 15-8" contains the gap bit. This bit is set TRUE when there is a qualification gap between the current sample and the previous sample.

RawWithTimestamp Binary Data Format			
A1 7 6 5 4 3 2 1 0	A0 7 6 5 4 3 2 1 0	D1 7 6 5 4 3 2 1 0	D0 7 6 5 4 3 2 1 0
C3 7 6 5 4 3 2 1 0	C2 7 6 5 4 3 2 1 0	A3 7 6 5 4 3 2 1 0	A2 7 6 5 4 3 2 1 0
Clock Quals (See ordering)	Reserved 15–8	Reserved 7–0	Timestamp 5 pad + 50–48
Timestamp 47–40	Timestamp 39–32	Timestamp 31–24	Timestamp 23–16
Timestamp 15–8	Timestamp 7–0		

Clock/Quals Byte							
7 Pad	6 Pad	5 Pad	4 Pad	3 CK3	2 CK2	1 CK1	0 CK0

Reserved 15-8 Byte							
7 Resv'd	6 Resv'd	5 Resv'd	4 Resv'd	3 Gap	2 Resv'd	1 Resv'd	0 Resv'd

RawWithTimestamp binary data format for TLA7N3 LA modules and the TLA6X3 logic analyzer

A main data sample for a stand-alone module consists of 22 bytes, starting with A3(7-0) and ending with Timestamp(7-0).

A main data sample for a merged pair consists of 22 bytes for the master module, starting with A3(7-0) and ending with Timestamp(7-0) plus an additional 13 bytes for the slave module, starting with A3(7-0) and ending with Clock/Quals. The byte labeled "Reserved 15-8" contains the gap bit. This bit is set TRUE when there is a qualification gap between the current sample and the previous sample.

RawWithTimestamp Binary Data Format			
A3 7 6 5 4 3 2 1 0	A2 7 6 5 4 3 2 1 0	D3 7 6 5 4 3 2 1 0	D2 7 6 5 4 3 2 1 0
A1 7 6 5 4 3 2 1 0	A0 7 6 5 4 3 2 1 0	D1 7 6 5 4 3 2 1 0	D0 7 6 5 4 3 2 1 0
C3 7 6 5 4 3 2 1 0	C2 7 6 5 4 3 2 1 0	C1 7 6 5 4 3 2 1 0	C0 7 6 5 4 3 2 1 0
Clock Quals (See ordering)	Reserved 15–8	Reserved 7–0	Timestamp 5 pad + 50–48
Timestamp 47–40	Timestamp 39–32	Timestamp 31–24	Timestamp 23–16
Timestamp 15–8	Timestamp 7–0		

Clock/Quals Byte							
7 Pad	6 Pad	5 Q1	4 Q0	3 CK3	2 CK2	1 CK1	0 CK0

Reserved 15-8 Byte							
7 Resv'd	6 Resv'd	5 Resv'd	4 Resv'd	3 Gap	2 Resv'd	1 Resv'd	0 Resv'd

RawWithTimestamp binary data format for TLA7N4/P4/Q4 LA modules and the TLA6X4 logic analyzer

A main data sample for a stand-alone module consists of 26 bytes, starting with E3(7-0) and ending with Timestamp(7-0).

A main data sample for a merged pair consists of 26 bytes for the master module, starting with E3(7-0) and ending with Timestamp(7-0) plus an additional 17 bytes for the slave module, starting with E3(7-0) and ending with Clock/Quals. The byte labeled "Reserved 15-8" contains the gap bit. This bit is set TRUE when there is a qualification gap between the current sample and the previous sample.

RawWithTimestamp Binary Data Format			
E3 7 6 5 4 3 2 1 0	E2 7 6 5 4 3 2 1 0	E1 7 6 5 4 3 2 1 0	E0 7 6 5 4 3 2 1 0
A3 7 6 5 4 3 2 1 0	A2 7 6 5 4 3 2 1 0	D3 7 6 5 4 3 2 1 0	D2 7 6 5 4 3 2 1 0
A1 7 6 5 4 3 2 1 0	A0 7 6 5 4 3 2 1 0	D1 7 6 5 4 3 2 1 0	D0 7 6 5 4 3 2 1 0
C3 7 6 5 4 3 2 1 0	C2 7 6 5 4 3 2 1 0	C1 7 6 5 4 3 2 1 0	C0 7 6 5 4 3 2 1 0
Clock Quals (See ordering)	Reserved 15–8	Reserved 7–0	Timestamp 5 pad + 50–48
Timestamp 47–40	Timestamp 39–32	Timestamp 31–24	Timestamp 23–16
Timestamp 15–8	Timestamp 7–0		

Clock/Quals Byte							
7 Q3	6 Q2	5 Q1	4 Q0	3 CK3	2 CK2	1 CK1	0 CK0

Reserved 15-8 Byte							
7 Resv'd	6 Resv'd	5 Resv'd	4 Resv'd	3 Gap	2 Resv'd	1 Resv'd	0 Resv'd

RawWithTimestamp binary data format for a merged LA module

A merged module combines data from the master and slave module(s) into each data sample. The data sample for a merged module consists of the data for the master module immediately followed by data from the slave module(s) as shown below:

Master > Slave 1 > Slave 2 > Slave 3 > Slave 4

Data for the slave module(s) never contain status or time stamp bytes.

For example, for two merged TLA7P4 modules, a main data sample consists of 43 bytes: 26 bytes of E3(7-0) through Timestamp(7-0) for the master module followed by 17 bytes of E3(7-0) through Clock/Quals for the slave module.

When using TLA7Axx Series Logic Analyzer modules, the data format for 4-way and 5-way merges is an extension of the pattern used for 3-way merges. The data for each successive slave is appended to the sample and the number of bytes per sample grows accordingly. For example, the binary data format used with five merged TLA7AB4 modules would have 94 bytes per sample and Slave 4 data would be contained in the last 17 bytes.

The following table shows a 2-way merge:

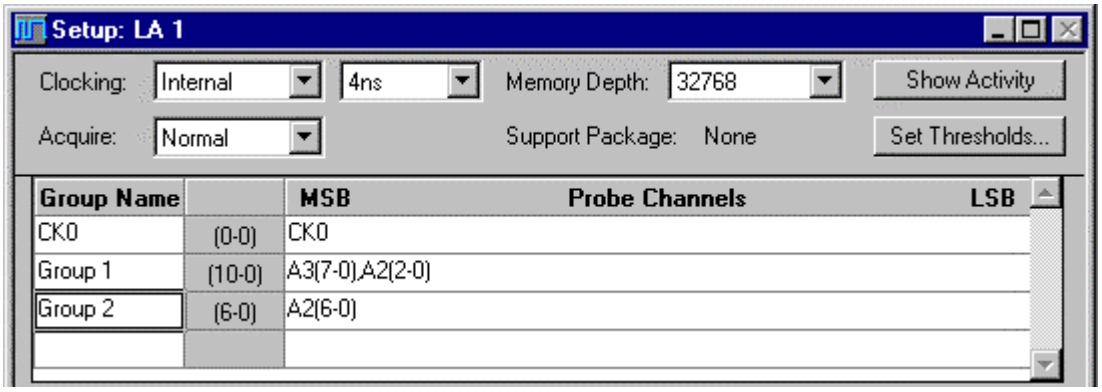
RawWithTimestamp Binary Data Format for a merged LA module			
E3 master 7 6 5 4 3 2 1 0	E2 master 7 6 5 4 3 2 1 0	E1 master 7 6 5 4 3 2 1 0	E0 master 7 6 5 4 3 2 1 0
A3 master 7 6 5 4 3 2 1 0	A2 master 7 6 5 4 3 2 1 0	D3 master 7 6 5 4 3 2 1 0	D2 master 7 6 5 4 3 2 1 0
A1 master 7 6 5 4 3 2 1 0	A0 master 7 6 5 4 3 2 1 0	D1 master 7 6 5 4 3 2 1 0	D0 master 7 6 5 4 3 2 1 0
C3 master 7 6 5 4 3 2 1 0	C2 master 7 6 5 4 3 2 1 0	C1 master 7 6 5 4 3 2 1 0	C0 master 7 6 5 4 3 2 1 0
Clock Quals master	Reserved 15–8	Reserved 7–0	Timestamp 5 pad + 50–48
Timestamp 47–40	Timestamp 39–32	Timestamp 31–24	Timestamp 23–16
Timestamp 15–8	Timestamp 7–0	E3 slave 1 7 6 5 4 3 2 1 0	E2 slave 1 7 6 5 4 3 2 1 0
E1 slave 1 7 6 5 4 3 2 1 0	E0 slave 1 7 6 5 4 3 2 1 0	A3 slave 1 7 6 5 4 3 2 1 0	A2 slave 1 7 6 5 4 3 2 1 0
D3 slave 1 7 6 5 4 3 2 1 0	D2 slave 1 7 6 5 4 3 2 1 0	A1 slave 1 7 6 5 4 3 2 1 0	A0 slave 1 7 6 5 4 3 2 1 0
D1 slave 1 7 6 5 4 3 2 1 0	D0 slave 1 7 6 5 4 3 2 1 0	C3 slave 1 7 6 5 4 3 2 1 0	C2 slave 1 7 6 5 4 3 2 1 0
C1 slave 1 7 6 5 4 3 2 1 0	C0 slave 1 7 6 5 4 3 2 1 0	Clock Quals slave 1	

The following table shows a 3-way merge:

RawWithTimestamp Binary Data Format for a merged LA module			
E3 master 7 6 5 4 3 2 1 0	E2 master 7 6 5 4 3 2 1 0	E1 master 7 6 5 4 3 2 1 0	E0 master 7 6 5 4 3 2 1 0
A3 master 7 6 5 4 3 2 1 0	A2 master 7 6 5 4 3 2 1 0	D3 master 7 6 5 4 3 2 1 0	D2 master 7 6 5 4 3 2 1 0
A1 master 7 6 5 4 3 2 1 0	A0 master 7 6 5 4 3 2 1 0	D1 master 7 6 5 4 3 2 1 0	D0 master 7 6 5 4 3 2 1 0
C3 master 7 6 5 4 3 2 1 0	C2 master 7 6 5 4 3 2 1 0	C1 master 7 6 5 4 3 2 1 0	C0 master 7 6 5 4 3 2 1 0
Clock Quals master	Reserved 15–8	Reserved 7–0	Timestamp 5 pad + 50–48
Timestamp 47–40	Timestamp 39–32	Timestamp 31–24	Timestamp 23–16
Timestamp 15–8	Timestamp 7–0	E3 slave 1 7 6 5 4 3 2 1 0	E2 slave 1 7 6 5 4 3 2 1 0
E1 slave 1 7 6 5 4 3 2 1 0	E0 slave 1 7 6 5 4 3 2 1 0	A3 slave 1 7 6 5 4 3 2 1 0	A2 slave 1 7 6 5 4 3 2 1 0
D3 slave 1 7 6 5 4 3 2 1 0	D2 slave 1 7 6 5 4 3 2 1 0	A1 slave 1 7 6 5 4 3 2 1 0	A0 slave 1 7 6 5 4 3 2 1 0
D1 slave 1 7 6 5 4 3 2 1 0	D0 slave 1 7 6 5 4 3 2 1 0	C3 slave 1 7 6 5 4 3 2 1 0	C2 slave 1 7 6 5 4 3 2 1 0
C1 slave 1 7 6 5 4 3 2 1 0	C0 slave 1 7 6 5 4 3 2 1 0	Clock Quals slave 1	E3 slave 2 7 6 5 4 3 2 1 0
E2 slave 2 7 6 5 4 3 2 1 0	E1 slave 2 7 6 5 4 3 2 1 0	E0 slave 2 7 6 5 4 3 2 1 0	A3 slave 2 7 6 5 4 3 2 1 0
A2 slave 2 7 6 5 4 3 2 1 0	D3 slave 2 7 6 5 4 3 2 1 0	D2 slave 2 7 6 5 4 3 2 1 0	A1 slave 2 7 6 5 4 3 2 1 0
A0 slave 2 7 6 5 4 3 2 1 0	D1 slave 2 7 6 5 4 3 2 1 0	D0 slave 2 7 6 5 4 3 2 1 0	C3 slave 2 7 6 5 4 3 2 1 0
C2 slave 2 7 6 5 4 3 2 1 0	C1 slave 2 7 6 5 4 3 2 1 0	C0 slave 2 7 6 5 4 3 2 1 0	Clock Quals slave 2

AllGroupsWithTimestamp binary data format for LA modules

As an example of an AllGroupsWithTimestamp binary data format, define groups as shown in the LA Setup window below.



In the above example, the data sample would have the following format, where X represents a pad bit.

AllGroupsWithTimestamp Data Sample			
C K 0 X X X X X X X 0	G r o u p 1 X X X X X 10 9 8	G r o u p 1 7 6 5 4 3 2 1 0	G r o u p 2 X 6 5 4 3 2 1 0
Timestamp 5 pad + 50–48	Timestamp 47–40	Timestamp 39–32	Timestamp 31–24
Timestamp 23–16	Timestamp 15–8	Timestamp 7–0	

RawWithoutTimestamp binary data format for TLA7N1 LA modules and the TLA6X1 logic analyzer

A main data sample for a stand-alone module consists of 7 bytes, starting with C3(7-0) and ending with Reserved(7-0). A MagniVu data sample for a stand-alone module consists of 5 bytes, starting with C3(7-0) and ending with Clock/Quals. The byte labeled "Reserved 15-8" contains the gap bit. This bit is set TRUE when there is a qualification gap between the current sample and the previous sample.

RawWithoutTimestamp Binary Data Format			
C3 7 6 5 4 3 2 1 0	C2 7 6 5 4 3 2 1 0	A3 7 6 5 4 3 2 1 0	A2 7 6 5 4 3 2 1 0
Clock Quals (See ordering)	Reserved 15–8	Reserved 7–0	

Clock/Quals Byte							
7 Pad	6 Pad	5 Pad	4 Pad	3 CK3	2 Pad	1 Pad	0 CK0

Reserved 15-8 Byte							
7 Resv'd	6 Resv'd	5 Resv'd	4 Resv'd	3 Gap	2 Resv'd	1 Resv'd	0 Resv'd

RawWithoutTimestamp binary data format for TLA7N2/P2/Q2 LA modules and the TLA6X2 logic analyzer

A main data sample for a stand-alone module consists of 11 bytes, starting with A1(7-0) and ending with Reserved(7-0). A MagniVu data sample for a stand-alone module consists of 9 bytes, starting with A1(7-0) and ending with Clock/Quals. The byte labeled "Reserved 15-8" contains the gap bit. This bit is set TRUE when there is a qualification gap between the current sample and the previous sample.

RawWithoutTimestamp Binary Data Format			
A1 7 6 5 4 3 2 1 0	A0 7 6 5 4 3 2 1 0	D1 7 6 5 4 3 2 1 0	D0 7 6 5 4 3 2 1 0
C3 7 6 5 4 3 2 1 0	C2 7 6 5 4 3 2 1 0	A3 7 6 5 4 3 2 1 0	A2 7 6 5 4 3 2 1 0
Clock Quals (See ordering)	Reserved 15–8	Reserved 7–0	

Clock/Quals Byte							
7 Pad	6 Pad	5 Pad	4 Pad	3 CK3	2 CK2	1 CK1	0 CK0

Reserved 15-8 Byte							
7 Resv'd	6 Resv'd	5 Resv'd	4 Resv'd	3 Gap	2 Resv'd	1 Resv'd	0 Resv'd

RawWithoutTimestamp binary data format for TLA7N3 LA modules and the TLA6X3 logic analyzer

A main data sample for a stand-alone module consists of 15 bytes, starting with A3(7-0) and ending with Reserved(7-0). A MagniVu data sample for a stand-alone module consists of 13 bytes, starting with A3(7-0) and ending with Clock/Quals. A data sample for a merged pair consists of the bytes for the master module plus the bytes for the slave module. The byte labeled "Reserved 15-8" contains the gap bit. This bit is set TRUE when there is a qualification gap between the current sample and the previous sample.

If you use the TLA7N3/TLA7N3 LA module as the master module of the merged pair, its data consists of 15 bytes (main) or 13 bytes (MagniVu) as described above.

RawWithoutTimestamp Binary Data Format			
A3 7 6 5 4 3 2 1 0	A2 7 6 5 4 3 2 1 0	D3 7 6 5 4 3 2 1 0	D2 7 6 5 4 3 2 1 0
A1 7 6 5 4 3 2 1 0	A0 7 6 5 4 3 2 1 0	D1 7 6 5 4 3 2 1 0	D0 7 6 5 4 3 2 1 0
C3 7 6 5 4 3 2 1 0	C2 7 6 5 4 3 2 1 0	C1 7 6 5 4 3 2 1 0	C0 7 6 5 4 3 2 1 0
Clock Quals (See ordering)	Reserved 15–8	Reserved 7–0	

Clock/Quals Byte							
7 Pad	6 Pad	5 Q1	4 Q0	3 CK3	2 CK2	1 CK1	0 CK0

Reserved 15-8 Byte							
7 Resv'd	6 Resv'd	5 Resv'd	4 Resv'd	3 Gap	2 Resv'd	1 Resv'd	0 Resv'd

RawWithoutTimestamp binary data format for TLA7N4/P4/Q4 LA modules and the TLA6X4 logic analyzer

A main data sample for a stand-alone module consists of 19 bytes, starting with E3(7-0) and ending with Reserved(7-0). A MagniVu data sample for a stand-alone module consists of 17 bytes, starting with E3(7-0) and ending with Clock/Quals. A data sample for a merged pair consists of the bytes for the master module plus the bytes for the slave module. The byte labeled "Reserved 15-8" contains the gap bit. This bit is set TRUE when there is a qualification gap between the current sample and the previous sample.

If you use the TLA7N4/P4/Q4 LA module as the master module of the merged pair, the data consists of 19 bytes (main) or 17 bytes (MagniVu) as described above.
If you use the TLA7P4/N4/Q4 LA module as the slave module of the merged pair, the data consists of 17 bytes, starting with E3(7-0) and ending with Clock/Quals.

RawWithoutTimestamp Binary Data Format

E3 7 6 5 4 3 2 1 0	E2 7 6 5 4 3 2 1 0	E1 7 6 5 4 3 2 1 0	E0 7 6 5 4 3 2 1 0
A3 7 6 5 4 3 2 1 0	A2 7 6 5 4 3 2 1 0	D3 7 6 5 4 3 2 1 0	D2 7 6 5 4 3 2 1 0
A1 7 6 5 4 3 2 1 0	A0 7 6 5 4 3 2 1 0	D1 7 6 5 4 3 2 1 0	D0 7 6 5 4 3 2 1 0
C3 7 6 5 4 3 2 1 0	C2 7 6 5 4 3 2 1 0	C1 7 6 5 4 3 2 1 0	C0 7 6 5 4 3 2 1 0
Clock Quals (See ordering)	Reserved 15–8	Reserved 7–0	

Clock/Quals Byte

7	6	5	4	3	2	1	0
Q3	Q2	Q1	Q0	CK3	CK2	CK1	CK0

Reserved 15-8 Byte

7	6	5	4	3	2	1	0
Resv'd	Resv'd	Resv'd	Resv'd	Gap	Resv'd	Resv'd	Resv'd

RawWithoutTimestamp binary data format for a merged LA module

This is the same as the RawWithTimestamp binary data format for a merged LA module except that all time stamp bytes are excluded.

For example, for two merged TLA7P4 modules, a main data sample consists of 36 bytes: 19 bytes of E3(7-0) through Reserved(7-0) for the master module followed by 17 bytes of E3(7-0) through Clock/Quals for the slave module.

When using TLA7Axx Series Logic Analyzer modules, the data format for 4-way and 5-way merges is an extension of the pattern used for 3-way merges. The data for each successive slave is appended to the sample and the number of bytes per sample grows accordingly. For example, the binary data format used with five merged TLA7AB4 modules would have 87 bytes per sample and Slave 4 data would be contained in the last 17 bytes.

The following table shows a 2-way merge.

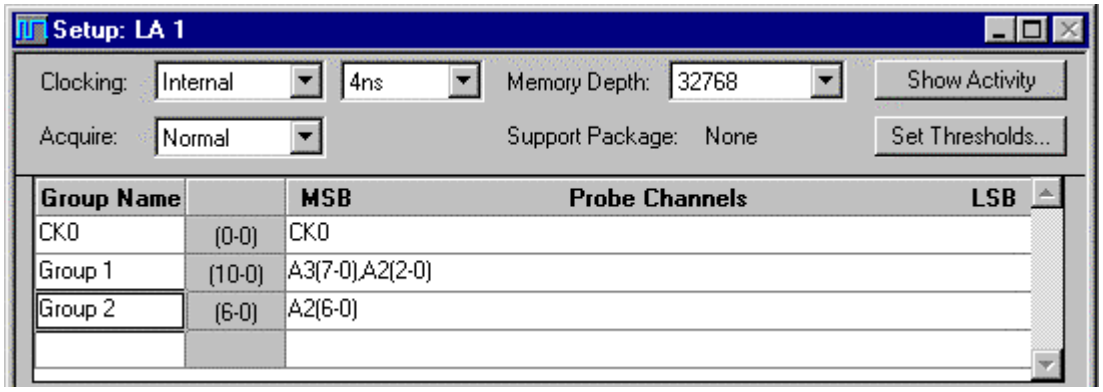
RawWithoutTimestamp Binary Data Format for a merged LA module			
E3 master 7 6 5 4 3 2 1 0	E2 master 7 6 5 4 3 2 1 0	E1 master 7 6 5 4 3 2 1 0	E0 master 7 6 5 4 3 2 1 0
A3 master 7 6 5 4 3 2 1 0	A2 master 7 6 5 4 3 2 1 0	D3 master 7 6 5 4 3 2 1 0	D2 master 7 6 5 4 3 2 1 0
A1 master 7 6 5 4 3 2 1 0	A0 master 7 6 5 4 3 2 1 0	D1 master 7 6 5 4 3 2 1 0	D0 master 7 6 5 4 3 2 1 0
C3 master 7 6 5 4 3 2 1 0	C2 master 7 6 5 4 3 2 1 0	C1 master 7 6 5 4 3 2 1 0	C0 master 7 6 5 4 3 2 1 0
Clock Quals master	Reserved 15–8	Reserved 7–0	E3 slave 1 7 6 5 4 3 2 1 0
E2 slave 1 7 6 5 4 3 2 1 0	E1 slave 1 7 6 5 4 3 2 1 0	E0 slave 1 7 6 5 4 3 2 1 0	A3 slave 1 7 6 5 4 3 2 1 0
A2 slave 1 7 6 5 4 3 2 1 0	D3 slave 1 7 6 5 4 3 2 1 0	D2 slave 1 7 6 5 4 3 2 1 0	A1 slave 1 7 6 5 4 3 2 1 0
A0 slave 1 7 6 5 4 3 2 1 0	D1 slave 1 7 6 5 4 3 2 1 0	D0 slave 1 7 6 5 4 3 2 1 0	C3 slave 1 7 6 5 4 3 2 1 0
C2 slave 1 7 6 5 4 3 2 1 0	C1 slave 1 7 6 5 4 3 2 1 0	C0 slave 1 7 6 5 4 3 2 1 0	Clock Quals slave 1

The following table shows a 3-way merge:

RawWithoutTimestamp Binary Data Format for a merged LA module			
E3 master 7 6 5 4 3 2 1 0	E2 master 7 6 5 4 3 2 1 0	E1 master 7 6 5 4 3 2 1 0	E0 master 7 6 5 4 3 2 1 0
A3 master 7 6 5 4 3 2 1 0	A2 master 7 6 5 4 3 2 1 0	D3 master 7 6 5 4 3 2 1 0	D2 master 7 6 5 4 3 2 1 0
A1 master 7 6 5 4 3 2 1 0	A0 master 7 6 5 4 3 2 1 0	D1 master 7 6 5 4 3 2 1 0	D0 master 7 6 5 4 3 2 1 0
C3 master 7 6 5 4 3 2 1 0	C2 master 7 6 5 4 3 2 1 0	C1 master 7 6 5 4 3 2 1 0	C0 master 7 6 5 4 3 2 1 0
Clock Quals master	Reserved 15–8	Reserved 7–0	E3 slave 1 7 6 5 4 3 2 1 0
E2 slave 1 7 6 5 4 3 2 1 0	E1 slave 1 7 6 5 4 3 2 1 0	E0 slave 1 7 6 5 4 3 2 1 0	A3 slave 1 7 6 5 4 3 2 1 0
A2 slave 1 7 6 5 4 3 2 1 0	D3 slave 1 7 6 5 4 3 2 1 0	D2 slave 1 7 6 5 4 3 2 1 0	A1 slave 1 7 6 5 4 3 2 1 0
A0 slave 1 7 6 5 4 3 2 1 0	D1 slave 1 7 6 5 4 3 2 1 0	D0 slave 1 7 6 5 4 3 2 1 0	C3 slave 1 7 6 5 4 3 2 1 0
C2 slave 1 7 6 5 4 3 2 1 0	C1 slave 1 7 6 5 4 3 2 1 0	C0 slave 1 7 6 5 4 3 2 1 0	Clock Quals slave 1
E3 slave 2 7 6 5 4 3 2 1 0	E2 slave 2 7 6 5 4 3 2 1 0	E1 slave 2 7 6 5 4 3 2 1 0	E0 slave 2 7 6 5 4 3 2 1 0
A3 slave 2 7 6 5 4 3 2 1 0	A2 slave 2 7 6 5 4 3 2 1 0	D3 slave 2 7 6 5 4 3 2 1 0	D2 slave 2 7 6 5 4 3 2 1 0
A1 slave 2 7 6 5 4 3 2 1 0	A0 slave 2 7 6 5 4 3 2 1 0	D1 slave 2 7 6 5 4 3 2 1 0	D0 slave 2 7 6 5 4 3 2 1 0
C3 slave 2 7 6 5 4 3 2 1 0	C2 slave 2 7 6 5 4 3 2 1 0	C1 slave 2 7 6 5 4 3 2 1 0	C0 slave 2 7 6 5 4 3 2 1 0
Clock Quals slave 2			

AllGroupsWithoutTimestamp binary data format for LA modules

As an example of AllGroupsWithoutTimestamp binary data format for LA modules, define groups as shown in the LA Setup window below.

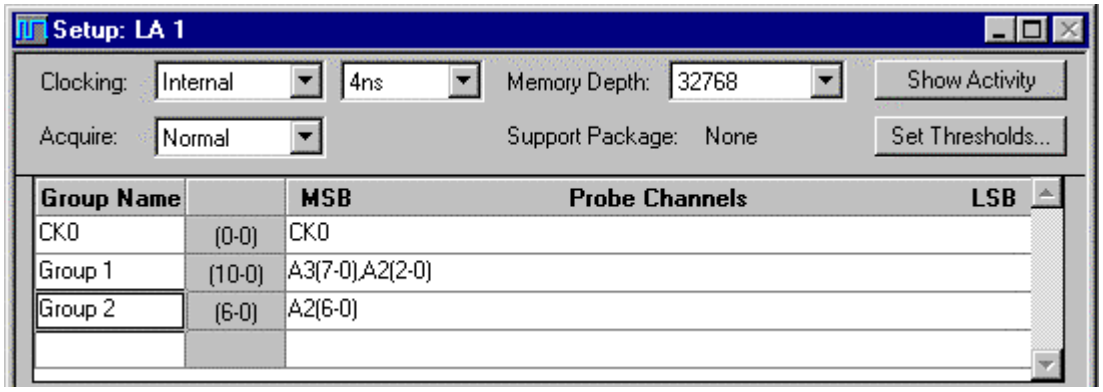


In the above example, a data sample would have the following format, where X represents a pad bit.

AllGroupsWithoutTimestamp Data Sample			
C K 0 X X X X X X X 0	G r o u p 1 X X X X X 10 9 8	G r o u p 1 7 6 5 4 3 2 1 0	G r o u p 2 X 6 5 4 3 2 1 0

GroupList binary data format for LA modules

As an example of the GroupList binary data format for LA modules, define groups as shown in the LA Setup window below.



If you specify the groups as follows,

"GroupList:Group2,Group1,CK0,Timestamp"

then, in the GroupList format, a data sample will have the following format, where X represents a pad bit.

GroupList Data Sample			
G r o u p 2 X 6 5 4 3 2 1 0	G r o u p 1 X X X X X 10 9 8	G r o u p 1 7 6 5 4 3 2 1 0	C K 0 X X X X X X X 0
Timestamp 5 pad + 50–48	Timestamp 47–40	Timestamp 39–32	Timestamp 31–24
Timestamp 23–16	Timestamp 15–8	Timestamp 7–0	

Binary data formats for DSO modules and external oscilloscopes

General characteristics

Delimiters are not used between samples in any of the binary data formats. Each sample is a fixed size.

A sample of a single DSO channel consists of a 16-bit short word in twos complement, signed form. This number is interpreted as a voltage using the offset and range settings of the channel (see IDSOModule::GetData or IDSOModule::ExportData). A zero value for the DSO data word maps to the offset voltage. The following is a summary of the values returned by the DSO:

Note: You must use IDSOModule::DefineDataFormat to specify a binary data format. You can then use IDSOModule::GetData or IDSOModule::ExportData to get the data in the specified format.

Values Returned by DSO		
Value		
MSB	LSB	Description
7F	FF	Overrange
7E	00	Top of requested range (equals offset voltage plus one-half of range)
00	02	. . .
00	01	. . .
00	00	Midpoint for signed numbers (equals offset voltage)
FF	FF	. . .
FF	FE	. . .
82	00	Bottom of requested range (equals offset voltage minus one-half range)
80	01	Under range
80	00	No samples taken or null point

AllChannels

In this data format, data for all channels is returned. The format of a sample for the various DSO modules follows:

TLA7D2/TLA7E2 and four channel external oscilloscopes. A sample contains four 16-bit short words, one for each of the four channels, beginning with Ch1 and ending with Ch4.

Ch1	Ch2	Ch3	Ch4
16-bit	16-bit	16-bit	16-bit

TLA7D1/TLA7E1 and two channel external oscilloscopes. A sample contains two 16-bit short words, one for each of the two channels, beginning with Ch1 and ending with Ch2.

Ch1	Ch2
16-bit	16-bit

ChannelList

In this data format, the specified channels are returned in the order specified. For example, suppose the user specified the channels as follows:

"ChannelList:2,1"

In this ChannelList format, a data sample would have the following format:

Ch2	Ch1
16-bit	16-bit

Text data formats for LA modules

General characteristics

Data is returned as an array of strings, one for each sample; each string is a fixed size.

Note: You must use ILAModule::DefineDataFormat to specify a text data format. You can then use ILAModule::GetData or ILAModule::ExportData to get the data in the specified format.

Violation data

Data formats for the violation data set are identical to those for the main data set. For the violation data set, a '1' in a bit position indicates a either a glitch or setup and hold violation occurred on the corresponding channel.

Time stamps

A time stamp value represents the time at which the sample was stored relative to the "start of the acquisition". It is important to note that the "start of the acquisition" is not the same as the first sample of data. The start of the acquisition is the moment at which the system is enabled to begin acquiring data. It takes some finite amount of time after the start of the acquisition before a module can actually store a sample of data (especially when using an external clock).

When acquisition data is returned in a raw format, time stamp values are returned in units of "ticks" where each tick represents a fixed amount of time. To obtain the time stamp values in picoseconds, it must be adjusted as follows:

*Time stamp value in ps =(Time stamp value in ticks *Time stamp Multiplier) +Expansion mainframe Offset +User-defined time alignment in ps*

The Time stamp Multiplier is obtained by calling ILAModule::GetTimestampMultiplier(). Expansion Mainframe Offset is 36000 ps if the module is in an expansion mainframe or 0 ps if it is not. User-defined time alignment is any time alignment defined for the module by the user.

When acquisition data is returned in a grouped format, time stamp values are returned in picoseconds.

RawWithTimestamp

This format is the same as the RawWithTimestamp binary data format for LA modules with the exception that the data is output as hexadecimal character digits instead of binary.

RawWithoutTimestamp

This format is the same as the RawWithoutTimestamp binary data format for LA modules with the exception that the data is output as hexadecimal character digits instead of binary.

AllGroupsWithTimestamp

In this data format, data for all channel groups is returned in the order in which the groups are defined in the LA Setup window. Time stamp data is also returned.

Groups are formatted in hexadecimal with the specified delimiter between groups. Channel groups in text format have leading zeros to form a constant-width field. Time stamp values in text format are in picoseconds with no units attached.

See AllGroupsWithTimestamp for an example of this data format.

AllGroupsWithoutTimestamp

In this data format, data for all channel groups is returned in the order in which the groups are defined in the LA Setup Window. No time stamp data is returned.

Groups are formatted in hexadecimal with the specified delimiter between groups. Channel groups in text format have leading zeros added to form a constant-width field.

See AllGroupsWithoutTimestamp for an example of this data format.

GroupList

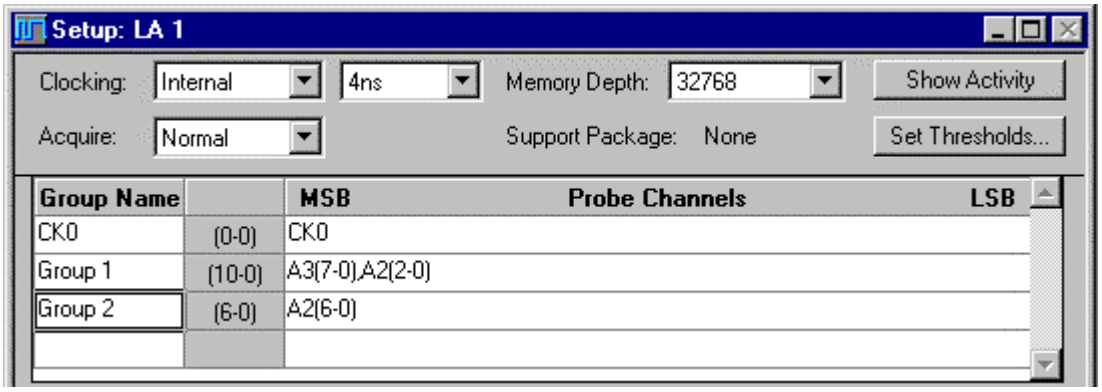
In this data format, data for specified channel groups is returned in the order in which the groups were specified.

Groups are formatted in hexadecimal with the specified delimiter between groups. Channel groups will have leading zeros added to form a constant-width field. Time stamp values in text format are in picoseconds and no units attached.

See GroupList for an example of this data format.

AllGroupsWithTimestamp text data format for LA modules

As an example of the AllGroupsWithTimestamp text data format for LA modules, define groups as shown in the LA Setup window.



In the AllGroupsWithTimestamp format with a space delimiter, a data sample will follow the format:

"0C 3FF 1 4000"

where 1 is the value of CK0

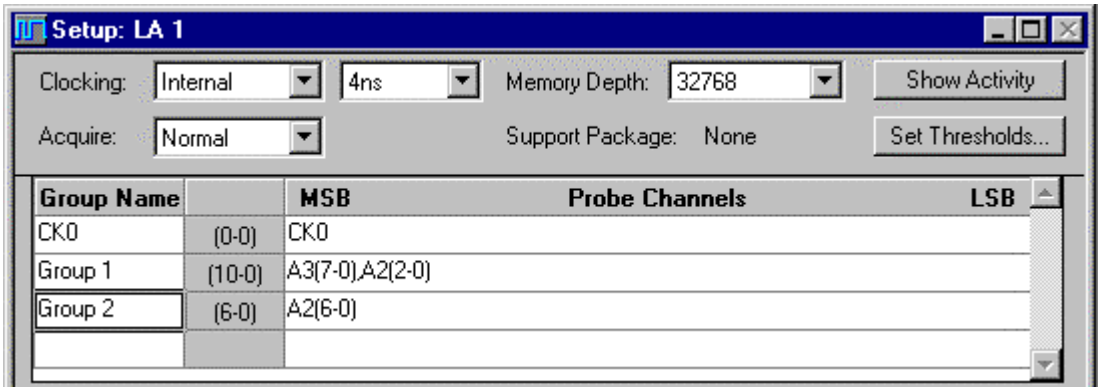
3FF is the value of Group 1

0C is the value of Group 2

4000 is the value of Time stamp in picoseconds from the start of the acquisition

AllGroupsWithoutTimestamp text data format for LA modules

In this example of the AllGroupsWithoutTimestamp text data format for LA modules, suppose that you defined groups as shown in the LA Setup window.



In the AllGroupsWithoutTimestamp format with a space delimiter, a data sample follows the format:

"0C 3FF 1"

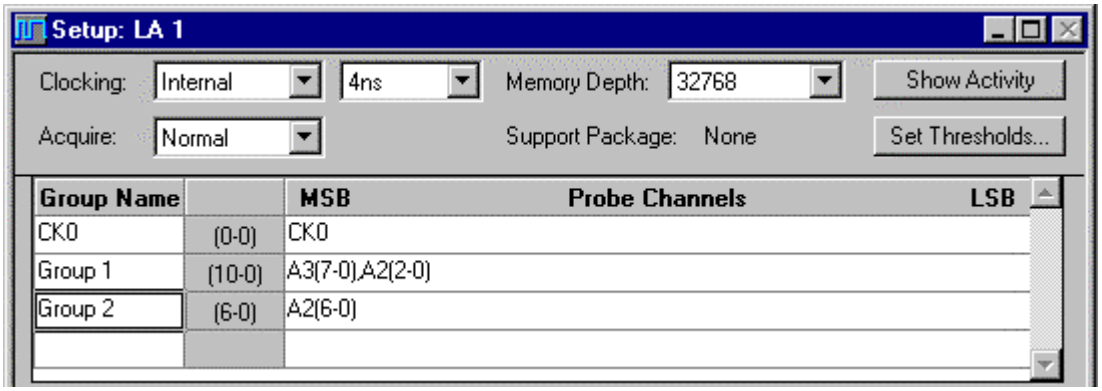
where 1 is the value of CK0

3FF is the value of Group 1

0C is the value of Group 2

GroupList text data format for LA modules

In this example of the GroupList text data format for LA modules, suppose that you defined the groups as shown in the LA Setup window.



If you specify the groups as follows:

"GroupList:Group 2,Group 1,CK0,Timestamp"

then, in the GroupList format with a space delimiter, a data sample would have the following format:

"0C 3FF 1 4000"

where 0C is the value of Group 2

3FF is the value of Group 1

1 is the value of CK0

4000 is the value of Time stamp in picoseconds from the start of the acquisition

Text data formats for DSO modules and external oscilloscopes

General characteristics

Data is returned in volts without the unit characters appended.

Note: You must use IDSOModule::DefineDataFormat to specify a text data format. You can then use IDSOModule::GetData or IDSOModule::ExportData to get the data in the specified format.

AllChannels

In this data format, data for all channels is returned. Samples for the various DSO modules follow:

TLA7D2/TLA7E2 and four channel external oscilloscopes

A sample contains data for each of the four channels, beginning with Ch1 and ending with Ch4.

Example:

"3.4444 3.555 3.444 3.555"

where 3.444 is the value of Ch1

3.555 is the value of Ch2

3.444 is the value of Ch3

3.555 is the value of Ch4

TLA7D1/TLA7E1 and two channel external oscilloscopes

A sample contains data for each of the two channels, beginning with Ch1 and ending with Ch2.

Example:

"3.444 3.555"

where 3.444 is the value of Ch1

3.555 is the value of Ch2

ChannelList

In this data format, data for the specified channels is returned in the order specified.

Example:

Suppose you specified the channels as follows:

"ChannelList:2,1"

In the ChannelList format, a data sample would have the following format:

"3.555 3.444"

where 3.444 is the value of Ch1

3.555 is the value of Ch2

Appendix

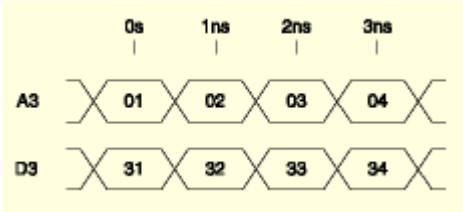
Internal 2X clocking mode

Internal 2X clocking mode acquisition allows data to be acquired and displayed at twice the normal maximum sample frequency of the logic analyzer. In this mode, half of the logic analyzer input channels are traded for twice the speed and twice the acquisition depth.

The following example illustrates how Internal 2X clocking acquisition differs from internal clocking acquisition and demonstrates when it can be used in observing fast data values.

Acquiring Data

The logic analyzer is set up to acquire data from a target system probed by the A-3(7-0) channels and the D3(7-0) channels. In the LA Setup dialog box, the A3(7-0) channels are used to form a group called "A3" and the D3(7-0) channels are used to form a group called "D3". The logic analyzer is set to trigger when Group A3 equals AA. Assume that the data present on the target system changes every clock cycle and has the following pattern:



The data is shown in hexadecimal radix and the times are in relation to the logic analyzer trigger position.

Note

The logic analyzer module used for this example has a minimum sample period of 2 ns for internal clocking mode.

First, an Internal 2X acquisition is taken. The clocking mode is set to Internal in the LA Setup dialog. The sample period is set to its minimum value of 2 ns. The acquisition samples stored in the logic analyzer module are as follows:

Internal clocking acquisition samples

A3 Channels	D3 Channels	Time
01	31	0 ns
03	33	2.0 ns

Note that the data on the target system was changing too fast for the logic analyzer to capture all the clock edges. Every other clock edge is missed. The timestamps stored in the acquisition samples are the times of the actual captured clock edges. Next, an internal 2X clocking mode acquisition is taken. The clocking mode is changed to Internal 2X in the LA Setup dialog box. The sample rate is now fixed at 1 ns. Also, each acquisition memory depth setting is doubled under internal 2X clocking mode.

In internal 2X clocking mode, the A3 channels are routed or demultiplexed into the D3 channels. The D3 channels actually see the physical signals coming from the A3 lines at the probe tip. This means that any physical signal connections to the D3 channel are ignored. The demultiplex occurs internal to the logic analyzer module, and does not require any external probing changes. With this arrangement, the A3 channels are referred to as the demultiplex source channels and the D3 channels are the demultiplex destination channels. Note in the LA Setup dialog box channel grid that all of the demultiplex destination

channels are highlighted in pink and demultiplex source channels are highlighted in blue when Internal 2X mode is selected.

For internal clocking acquisitions, the A3 channels in the acquisition sample are used to store data from the A3 probe inputs while the D3 channels in the acquisition sample are used to store data from the D3 probe units. In internal 2X clocking mode, the A3 channels store data from the A3 probe inputs at time t, while the D3 channels store data from the A3 probe inputs at time t + 1 ns. There are two logical acquisition samples stored in each physical sample. The acquisition samples stored in the LA module in internal 2X clocking mode are as follows:

Internal 2X clocking acquisition samples

A3 Channels	D3 Channels	Time
01	02	1.0 ns
03	04	3.0 ns

TPI Data

When retrieving internal clocking data through TPI, the data samples returned by ILAModule::GetData track the stored acquisition data closely. The data samples for the internal clocking acquisition above might look like the following:

Internal clocking TPI data

A3 Channels	D3 Channels	Timestamp
01	31	0 ns
03	33	2.0 ns

In Internal 2X clocking mode, there are twice as many samples returned by ILAModule::GetData as there are physically stored on the acquisition card. Each physical sample is expanded into its two component logical samples, which are returned by GetData.

Internal 2X clocking TPIdata

A3 Channels	D3 Channels	Timestamp
01	02	0 ns
02	02	1 ns
03	04	2 ns
04	04	3 ns

Looking at the A3 channels in the TPI data, you can see that the data pattern matches the behavior of the target system from which the data was acquired. In this window, the first and third samples come directly from the stored acquisition, while second and fourth samples are synthesized from the demultiplex destination data. In order to form the synthesized data samples, the data from each demultiplex destination channel is copied into its demultiplex source channel.

The demultiplex source and destination channel mappings used for internal 2X clocking mode depend on the logic analyzer module width. For more information about demultiplex channel mappings, see Probe Demultiplexing.

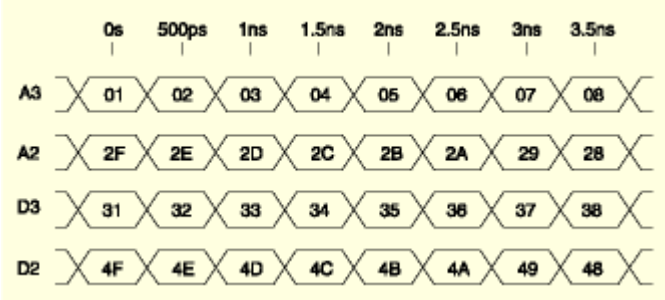
Internal 4X clocking mode

Internal 4X mode acquisitions allow data to be acquired and displayed at four times the normal maximum sample frequency of the logic analyzer. In this mode, three quarters of the logic analyzer input channels are traded for four times the speed and four times the acquisition depth.

The following example illustrates how an Internal 4X acquisition differs from normal 1X acquisition and demonstrates when it can be used in observing fast data transitions.

Acquiring Data

The logic analyzer is set up to acquire data from a target system probed by the A3(7-0), A2(7-0), D3(7-0) and D2(7-0) channels. In the LA Setup dialog, the A3(7-0) channels are used to form a group called "A3," the A2(7-0) channels are used to form a group called “A2, the D3(7-0) channels are used to form a group called "D3," and the D2(7-0) channels are used to form a group called “D2. Under Internal 4X clocking, data in the A3 channel is distributed to the A2, D3 and D2 channels. The logic analyzer is set to trigger when Group A3 equals AA. Assume that the data present on the target system changes every 500 ps and has the following pattern:



The data is shown in hexadecimal radix and the times are in relation to the logic analyzer trigger position.

Note

The logic analyzer module used for this example has a minimum sample period of 2 ns for Internal clocking mode

First, an internal clocking acquisition is taken. The clocking mode is set to Internal in the LA Setup dialog. The sample period is set to 2 ns. In internal clocking mode, no demultiplexing occurs; each channel receives its data directly from the probe tip. The acquisition samples stored in the logic analyzer module are as follows:

Internal clocking acquisition samples

A3 Channels	A2 Channels	D3 Channels	D2 Channels	Time
01	2F	31	4F	0 ns
05	2B	35	4B	2.0 ns

Note that the data on the target system was changing too fast for the logic analyzer to capture all the values.

Next, an internal 4X clocking mode acquisition is taken. The clocking mode is changed to Internal 4X in the LA Setup dialog. The sample rate is now fixed at 500 ps. Also, each acquisition memory depth setting is quadrupled under 4X mode.

In Internal 4X clocking mode, the A3 channels are routed or demultiplexed into the A2, D3 and D2 channels. The A2, D3 and D2 channels actually see the physical signals coming from the A3 lines. This means that any physical signal connections to the A2, D3 and D2 channels are ignored. The demultiplexing occurs internal to the logic analyzer module and does not require any external probing changes. With this arrangement, the A3 channels are referred to as the demultiplex source channels and the A2, D3 and D2 channels are the demultiplex destination channels. Note in the LA Setup dialog channel grid that all of the demultiplex source and destination channels are highlighted when internal 4X mode is selected.

For normal acquisitions, the A3 channels in the acquisition sample are used to store data from the A3 probe inputs, the A2 channels in the acquisition sample are used to store data from the A2 probe inputs, the D3 channels in the acquisition sample are used to store data from the D3 probe units, and the D2 channels in the acquisition sample are used to store data from the D2 probe units. In 4X mode, data

acquired from the A3 source channel is distributed to the A2, D3 and D2 destination channels. The A3 channels store data from the A3 probe inputs at time t, while the A2 channels store data from the A3 probe inputs at time t + 500 ps, the D3 channels store data from the A3 probe inputs at time t + 1 ns, and the D2 channels store data from the A3 probe inputs at time t + 1.5 ns. There are four logical acquisition samples stored in each physical sample. The acquisition samples stored in the LA module in 4X mode are as follows:

Internal 4X clocking acquisition samples

A3 Channels	A2 Channels	D3 Channels	D2 Channels	Time
01	02	03	04	1.0 ns
05	06	07	08	3.0 ns

TPI Data

When retrieving internal clocking data through TPI, the data samples returned by ILAModule::GetData track the stored acquisition data closely. The data samples for the internal clocking acquisition above might look like the following:

Internal clocking TPI data

A3 Channels	A2 Channels	D3 Channels	D2 Channels	Timestamp
01	2F	31	4F	0 ps
05	2B	35	4B	2.0 ns

In internal 4X clocking mode, there are four times as many samples returned by ILAModule::GetData as there are physically stored on the acquisition card. Each physical sample is expanded into its four component logical samples, which are returned by GetData.

Internal 4X clocking TPI data

A3 Channels	A2 Channels	D3 Channels	D2 Channels	Timestamp
01	02	03	04	0 ps
02	02	03	04	500 ps
03	02	03	04	1.0 ns
04	02	03	04	1.5 ns
05	06	07	08	2.0 ns
06	06	07	08	2.5 ns
07	06	07	08	3.0 ns
08	06	07	08	3.5 ns

Looking at the A3 channels in the TPI data, you can see that the data pattern matches the behavior of the target system from which the data was acquired. In this window, the first and fifth samples come directly from the stored acquisition, while the other samples are synthesized from the demultiplex destination data. In order to form the synthesized data samples, the data from each demultiplex destination channel is copied into its demultiplex source channel.

The demultiplex source and destination channel mappings used for internal 4X clocking mode depend on the logic analyzer module width. For more information about demultiplex channel mappings, see Probe Demultiplexing.

External 4X clocking mode

External 4X mode acquisitions allow data to be acquired and displayed at approximately three to four times the normal maximum sample frequency of the logic analyzer. In this mode, three quarters of the logic analyzer input channels are traded for three to four times the speed and four times the acquisition

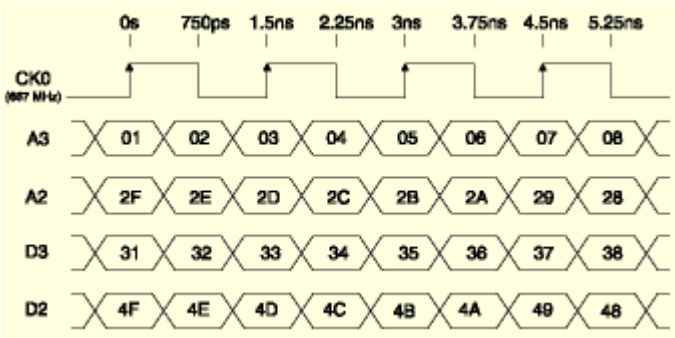
depth. The reason that the External 4X clocking mode data rate does not always increase by four times above the base synchronous rate of the logic analyzer is that the Setup/Hold window requirements of the logic analyzer may come into play. Setup and hold requirements on the acquisition data can limit the acquisition rate to a lesser value than if it were constrained by the maximum clock speed alone.

External 4X clocking mode is a double data rate (DDR) clocking mode, which means that the two acquisition samples are taken per clock cycle. This means that the data rate of the acquisition is double the rate of the clock signal that is driving it.

The following example illustrates how external 4X acquisition differs from external clocking acquisitions and demonstrates how external 4X clocking can be useful in capturing data from target systems with higher data rates than the base synchronous speed of the logic analyzer module.

Acquiring Data

The logic analyzer is set up to acquire data from a target system probed by the A3(7-0), A2(7-0), D3(7-0) and D2(7-0) channels. In the LA Setup dialog, the A3(7-0) channels are used to form a group called "A3," the A2(7-0) channels are used to form a group called “A2, the D3(7-0) channels are used to form a group called "D3," and the D2(7-0) channels are used to form a group called “D2. The logic analyzer is set to trigger when Group A3 equals 01. Assume that the data present on the target system changes twice every clock cycle and has the following pattern.



The data is shown in hexadecimal radix and the times are in relation to the logic analyzer trigger position.

For this example, assume that the logic analyzer module has a maximum synchronous acquisition speed of 450 MHz. This means that there is a minimum time of 2.22 ns between successive clock edges than can be captured in external clocking mode. The clock is running at a speed of 667 MHz, which only leaves 1.5 ns between successive clock edges. In external 4X mode, the logic analyzer is capable of capturing every rising or every falling edge of the 667 MHz clock and also capturing an additional sample between clock edges.

First, an external clocking acquisition is taken. The clocking mode is set to External in the LA Setup dialog box. The clock definition is set to the rising edge of CK0 in the External Clocking dialog box. The acquisition samples stored in the logic analyzer module are as follows:

External clocking acquisition samples

A3	A2	D3	D2	Timestamp
01	2F	31	4F	0 ns
05	2B	35	4B	3 ns

Note that the data on the target system was changing too fast for the logic analyzer to capture all the clock edges. Every other clock edge is missed. The timestamps stored in the acquisition samples are the times of the actual captured clock edges.

Next, an external 4X clocking mode acquisition is taken. The clocking mode is changed to External 4X in the LA Setup dialog box. The clock edge is set to the rising edge of CK0 in the External 4X Clocking dialog box. The Setup/Hold windows for all groups are left at their default settings, which means that the

data is sampled just before the rising clock edge. The Second Edge Delay for each group is set to 750 ps to take the second data sample just before the falling clock edge. Note that the logic analyzer does not capture the falling clock edge.

In External 4X clocking mode, the A3 channels are routed or demultiplexed into the A2, D3 and D2 channels. The A2, D3 and D2 channels actually see the physical signals coming from the A3 lines at the probe tip. This means that any physical signal connections to the A2, D3 and D2 channels are ignored. The demultiplexing occurs internal to the logic analyzer module and does not require any external probing changes. With this arrangement, the A3 channels are referred to as the demultiplex source channels and the A2, D3 and D2 channels are the demultiplex destination channels. Note in the LA Setup dialog channel grid that all of the demultiplex destination channels are highlighted in pink and the demultiplex source channels are highlighted in blue when external 4X mode is selected.

For external clocking acquisitions, the A3 channels in the acquisition sample are used to store data from the A3 probe inputs when a clock edge occurs; the A2 channels in the acquisition sample are used to store data from the A2 probe inputs; the D3 channels in the acquisition sample are used to store data from the D3 probe units; and the D2 channels in the acquisition sample are used to store data from the D2 probe units.

In external 4X clocking mode, the A3 channels store data from the A3 probe inputs when the first clock edge occurs; the A2 channels store data from the A3 probe inputs 750 ps after the first clock edge; the D3 channels store data from the A3 probe inputs when the second clock edge occurs; and the D2 channels store data from the A3 probe inputs 750 ps after the second clock edge. Each set of two subsequent clock edges follows the same pattern: A3 stores the data present at the first edge while A2 stores the data 750 ps later. Then D3 stores the data at the next rising clock edge while D2 stores the data 750 ps later. There are four logical acquisition samples stored in each physical sample. The acquisition samples stored in the LA module in external 4X clocking mode are as follows:

External 4X clocking acquisition samples

A3	A2	D3	D2	Timestamp
01	02	03	04	1.5 ns
05	06	07	08	4.5 ns

The timestamp values stored in the acquisition samples are actually the times of the second clock edges associated with the data that was saved in the second demultiplexed destination (D3) channels.

TPI Data

When retrieving external clocking data through TPI, the data samples returned by ILAModule::GetData track the stored acquisition data closely. The data samples for the external clocking acquisition above might look like the following:

External clocking TPI data

A3 Channels	A2 Channels	D3 Channels	D2 Channels	Timestamp
01	2F	31	4F	0 ns
05	2B	35	4B	3 ns

In external 4X clocking mode, there are four times as many samples returned by ILAModule::GetData as there are physically stored on the acquisition card. Each physical sample is expanded into its four component logical samples, which are returned by GetData.

External 4X clocking TPI data

A3	A2	D3	D2	Timestamp
01	02	03	04	0.5 ns
02	02	03	04	1.0 ns
03	02	03	04	1.5 ns
04	02	03	04	2.0 ns
05	06	07	08	3.5 ns

06	06	07	08	4.0 ns
07	06	07	08	4.5 ns
08	06	07	08	5.0 ns

Looking at the A3 channels in the TPI data, you can see that the data pattern matches the behavior of the target system from which the data was acquired. In this window, the first and fifth samples come directly from the stored acquisition, while the other samples are synthesized from the demultiplex destination data. In order to form the synthesized data samples, the data from each demultiplex destination channel is copied into its demultiplex source channel.

In the Timestamp column, the timestamps for the second generated samples come directly from the timestamps stored in the acquisition records. The timestamps for the other samples are not stored anywhere in the acquisition record. These timestamps are generated by offsetting them from the stored timestamp. The timestamps for the first and fifth samples are generated by subtracting 1 ns from the stored timestamp. The timestamps for the third and seventh samples are generated by subtracting 500 ps from the stored timestamp. And the timestamps for the fourth and eighth samples were generated by adding 500 ps to the stored timestamp. Generated timestamps to not show exactly where the data was sampled, but they are reasonably close for clock speeds approaching 667 MHz.

The demultiplex source and destination channel mappings used for external 4X clocking mode depend on the logic analyzer module width. For more information about demultiplex channel mappings, see Probe Demultiplexing.

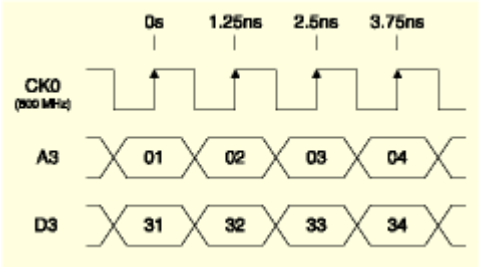
External 2X clocking mode

External 2X clocking mode allows data to be acquired and displayed at twice the normal maximum sample frequency of the logic analyzer. In this mode, half of the logic analyzer input channels are traded for twice the speed and twice the acquisition depth.

The following example illustrates how external 2X clocking acquisition differs from external clocking acquisitions and demonstrates how external 2X clocking can be useful in capturing data from target systems with higher clock frequencies than the base synchronous speed of the logical analyzer module.

Acquiring Data

The logic analyzer is set up to acquire data from a target system probed by the A-3(7-0) channels and the D3(7-0) channels. In the LA Setup dialog, the A3(7-0) channels are used to form a group called "A3" and the D3(7-0) channels are used to form a group called "D3". The logic analyzer is set to trigger when Group A3 equals 01. Assume that the data present on the target system changes every clock cycle and has the following pattern:



The data is shown in hexadecimal radix and the times are in relation to the logic analyzer trigger position.

For example, assume that the logic analyzer module has a maximum synchronous acquisition speed of 450 MHz. This means that there is a minimum time of 2.22 ns between successive clock edges that can be captured in external clocking mode. The clock is running at a speed of 800 MHz, which only leaves 1.25

ns between successive rising clock edges. In external 2X clocking mode, the logic analyzer is capable of capturing every rising or every falling edge of the 800 MHz clock.

First, an external clocking acquisition is taken. The clocking mode is set to External in the LA Setup dialog box. The clock definition is set to the rising edge of CK0 in the External Clocking dialog box. The acquisition samples stored in the logic analyzer module are as follows:

External clocking acquisition samples

A3 Channels	D3 Channels	Time
01	31	0 ns
03	33	2.5 ns

Note that the data on the target system was changing too fast for the logic analyzer to capture all the clock edges. Every other clock edge is missed. The timestamps stored in the acquisition samples are the times of the actual captured clock edges.

Next, an external 2X clocking mode acquisition is taken. The clocking mode is changed to External 2X in the LA Setup dialog box. The clock edge is set to the rising edge of CK0 in the External 2X Clocking dialog box. The Setup/Hold windows for all groups are left at their default settings, which means that the data is sampled just before the rising clock edge.

In external 2X clocking mode, the A3 channels are routed or demultiplexed into the D3 channels. The D3 channels actually see the signals coming from the A3 lines at the probe tip. This means that any physical signal connections to the D3 channel are ignored. The demultiplexing occurs internal to the logic analyzer module, and does not require any external probing changes. With this arrangement, the A3 channels are referred to as the demultiplex source channels and the D3 channels are the demultiplex destination channels. Note in the LA Setup dialog channel grid that all of the demultiplex destination channels are highlighted in pink and the demultiplex source channels are highlighted in blue when external 2X mode is selected.

For external clocking acquisitions, the A3 channels in the acquisition sample are used to store data from the A3 probe inputs when a clock edge occurs, while the D3 channels in the acquisition sample are used to store data from the D3 probe units. In external 2X mode, the A3 channels store data from the A3 probe inputs when the first clock edge occurs, while the D3 channels store data from the A3 probe inputs when the second clock edge occurs. Each set of two subsequent clock edges follows the same pattern: A3 stores the data present at the first edge, while D3 stores the data at the following edge. There are two logical acquisition samples stored in each physical sample. The acquisition samples stored in the LA module in external 2X clocking mode are as follows:

External 2X clocking acquisition samples

A3 Channels	D3 Channels	Time
01	02	1.25 ns
03	04	3.75 ns

The timestamp values stored in the acquisition samples are actually the times of the second clock edges associated with the data that was saved in the demultiplex destination (D3) channels.

TPI Data

When retrieving external clocking data through TPI, the data samples returned by ILAModule::GetData track the stored acquisition data closely. The data samples for the external clocking acquisition above might look like the following:

External clocking TPI data

A3 Channels	D3 Channels	Timestamp
01	31	0 ns
03	33	2.5 ns

In external 2X clocking mode, there are twice as many samples returned by ILAModule::GetData as there are physically stored on the acquisition card. Each physical sample is expanded into its two component logical samples, which are returned by GetData.

External 2X clocking TPI data

A3 Channels	D3 Channels	Timestamp
01	02	0.25 ns
02	02	1.25 ns
03	04	2.75 ns
04	04	3.75 ns

Looking at the A3 column in the Listing window, you can see that the data pattern matches the behavior of the target system from which the data was acquired. In this window, the first and third samples come directly from the stored acquisition, while the second and third samples are synthesized from the demultiplex destination data. In order to form the synthesized data samples, the data from each demultiplex destination channel is copied into its demultiplex source channel.

In the Timestamp column, the timestamps for the generated samples (second and fourth samples) come directly from the timestamps stored in the acquisition records. The timestamps for the first and third samples are not stored anywhere in the acquisition record, so they are generated by subtracting 1ns from the stored timestamp. Generated timestamps do not show exactly where the data was sampled, but they are reasonably close for clock speeds approaching 800MHz.

The demultiplex source and destination channel mappings used for external 2X clocking mode depend on the logic analyzer module width. For more information about demultiplex channel mappings, see Probe demultiplexing.

Probe demultiplexing

The channels available for demultiplexing and the demultiplex mapping depend on the width of the module and the type of demultiplexing you select. Mappings for each module type are shown in the following tables.

2X demultiplexing

The following table contains the probe channels available for 2X demultiplexing, depending on the width of the module.

136 Channel Module	102 Channel Module	68 Channel Module	34 Channel Module
A3 –> D3	A3 –> D3	A3 –> C3	A3 –> C3
A2 –> D2	A2 –> D2	A2 –> C2	A2 –> C2
A1 –> D1	A1 –> D1	A1 –> D1	
A0 –> D0	A0 –> D0	A0 –> D0	
C3 –> C1	C3 –> C1		
C2 –> C0	C2 –> C0		
E3 –> E1	CK0 –> Q1		
E2 –> E0	CK1 –> Q0		
CK2 –> Q3			
CK3 –> Q2			
CK0 –> Q1			
CK1 –> Q0			

4X demultiplexing

The following table contains the probe channels available for 4X demultiplexing, depending on the width of the module.

136 Channel Module	102 Channel Module	68 Channel Module	34 Channel Module
A1 –> A0, D1, D0	A3 –> A2, D3, D2	C3 –> C2, A3, A2	C3 –> C2, A3, A2
A3 –> A2, D3, D2	A1 –> A0, D1, D0	A1 –> A0, D1, D0	
C3 –> C2, C1, C0	C3 –> C2, C1, C0		
E3 –> E2, E1, E0	CK1 –> CK0, Q1, Q0		
CK3 –> CK2, Q3, Q2			
CK1 –> CK0, Q1, Q0			

Note

To increase the availability of clocks for driving data, clock and qualifier channels do not demultiplex in External 2X or External 4X clocking mode.

The Ax, Cx, Dx and Ex designations in the above tables represent blocks of eight data channels, while the CKx and Qx designations represent single clock and qualifier channels. The channel to the left of the arrow is the demultiplex source, while the channel(s) to the right of the arrow are the demultiplex destination. The source and destination channels form a demultiplex set.

Glossary

Glossary

Begin time

Begin time is the time stamp of the first sample of the module acquired data relative to the start of the acquisition.

Client

The client is an application that you write to control the TLA through TPI. The machine on which you run the client is called the client machine.

COM

The Component Object Model (COM) is a software architecture that allows applications to be built from binary software components. See **www.microsoft.com** for more information.

DCOM

The Distributed Component Object Model (DCOM) enables software components to communicate directly over a network in a reliable, secure, and efficient manner. See **www.microsoft.com** for more information.

End time

End time is the time stamp of the last sample of the module acquired data relative to the start of the acquisition.

External Oscilloscope

The TLA Application is able to utilize selected Tektronix TDS series oscilloscopes as an external data source. Data from an External Oscilloscope is automatically time correlated with internal modules and can be displayed in the standard TLA data windows.

Logical Module

A logical module can be a single hardware module, or merged modules as shown in the System window.

Main data sample

A main data sample refers to the standard LA module acquisition data as opposed to MagniVu or Glitch data.

Master module

The master module refers to the module that is to the left of a merged pair, or in the center of three merged modules.

Modal dialog

A modal dialog is one that must be dismissed before you can interact with the rest of the application.

If you have a modal dialog open, TPI disallows selected methods or property setting that would alter the system state.

Physical module

A physical module is a hardware module, such as an LA module that is installed in the mainframe.

Server

The TLA software application is called the server.

Share-level Access

Share-level access allows a password to be assigned to each shared resource. For Microsoft Windows 98-only networks, share-level access is the only option.

Slave module

The slave module refers to the module that is to the right of two merged modules, or on each side of the master module when three modules are merged.

User-level Access

User-level access allows a group of users to have access to each shared resource.

Version History

TPI Version History

In TPI Version 4.2

Added support for TLA7Axx Series Logic Analyzer modules and minor bug fixes for TPI Version 4.1.

In Version 4.1

New Properties and Methods:

Added support for External Oscilloscopes. Added and changed several new properties and methods. Refer to the following information for details:

ISystem::ExternalSignalIn – Specifies which Internal signal should be connected to External In or get current setting for External Signal In.

ISystem::ExternalSignalOut – Specifies which Internal signal should be connected to External Signal Out or get current setting for External Signal Out.

ISystem::ExternalSignalOutLowTrue – Set the logical polarity for External Signal Out.

ISystem::GetModuleSlotByName – Returns slot number with specified name.

ISystem::GetRepetitiveStopReason – Determines why the last repetitive acquisition stopped.

ILAModule::GetChannelName – Used to get the user assigned name for a channel of this module.

ILAModule::SetChannelName – Set assigned name for a channel of this module.

ILAModule::GetChanneGroup – Used to get a list of channels assigned to a channel group.

ILAModule::SetChanneGroup – Used to set the channel list for a channel group, or create a new channel group and assign channels to this list.

ILAModule::DeleteChanneGroup – Used to delete a channel group.

ILAModule::MemoryDepth – Used to get or set Memory Depth for this module.

Changed Methods

Each of the following methods will now return an error response (TLA700_E_MODAL_DIALOG_OPEN) if they are invoked while a Modal Dialog is open in the TLA application:

ISystem::LoadSystem

ISystem::Run

ISystem::Stop

ISystem::DefineRangeSymbolOptions

ISystem::LoadSymbolFile

ILAModule::LoadModule

ILAModule::LoadTrigger

ILAModule::SetEventValue

ILAModule::SetTriggerPosition

ILAModule::SetChannelName

ILAModule::SetChannelGroup

ILAModule::DeleteChannelGroup

IDSOModule::LoadModule

In the method ISystem::DefineRangeSymbolOptions, the parameter "MaxSymbols" is no longer used. There is no longer a fixed upper limit to the number of symbols that will be read from an object file. In order to preserve the interface for existing clients, the "MaxSymbols" parameter was replaced with a new parameter of the same type, named "Reserved". The value of the "Reserved" parameter is ignored.

All IDSOModule methods and properties now support External Oscilloscopes modules as well as Internal DSO modules.

Changed Properties

Each of the following properties will now return an error response (TLA700_E_MODAL_DIALOG_OPEN) if an attempt is made to set the value of the property while a Modal Dialog is open in the TLA application. Getting the value of the property is still allowed while a Modal Dialog is open:

ISystem::ExternalSignalIn

ISystem::ExternalSignalOut.

ISystem::ExternalSignalOutLowTrue

ISystem::Repetitive

ILAModule::Enabled

ILAModule::Name

ILAModule::MemoryDepth

IDSOModule::Enabled

IDSOModule::Name

In TPI Version 4.0

Added support for Windows 2000. Added a PDF file of the TLA Programmatic Interface (TPI) Online Help so that you can print out the help information, if desired.

In TPI Version 3.2

Added TLA TPI support for the TLA600 series of logic analyzers.

In TPI Version 3.1

Added several new methods and changed several existing methods. Refer to the following information for details.

Note: Microsoft Visual Basic users—if you use the Tektronix TLA Type Library, it will be marked "Missing" in the Project/References dialog. Clear the selection and click OK to exit the dialog. Reenter the Project/References dialog and recheck the check box labeled Tektronix TLA Type Library. Click OK to apply changes.

New Methods

ISystem::Repetitive

ISystem::RunCount

ISystem::GetModuleNames

ISystem::DefineRangeSymbolOptions

ISystem::LoadSymbolFile

ILAModule::Name

ILAModule::Enabled

ILAModule::GetStartTime

ILAModule::GetGroupNames

ILAModule::GetGroupSize

IDSOModule::Name

IDSOModule::Enabled

IDSOModule::GetStartTime

Changed Methods

The TLA server now supports multiple clients at the same time.

IApplication::ShowWindow - The application window is displayed by default when a client connects; the application window is also displayed when all clients have disconnected.

These two methods now include the first visible slot in the mainframes:

ISystem::GetNumModuleSlots - Returns a value of four for TLA704/TLA714 and a value of 13 for TLA711/TLA720.

ISystem::GetFirstModuleSlot - Returns a value of one for TLA704/TLA714 and a value of zero for TLA711/TLA720.

Expansion mainframes are supported in TPI. Extended slot numbers may be used to specify slot numbers in the expansion mainframes for the following methods:

ISystem::GetModuleTypesBySlot - Supports Expansion Interface modules.

ISystem::GetModulePropertiesBySlot - Supports Expansion Interface modules.

ISystem::GetModuleBySlot

Sample Suppression is now supported in LA modules:

ISystem::SaveSystem, ILAModule::SaveModule - The user now has the option of saving unsuppressed acquisition data.

ILAModule::GetNumSamples - If sample suppression is used, the value returned reflects the number of unsuppressed acquisition data samples.

ILAModule::GetTriggerSample - If the sample suppression is used, this method returns the adjusted position of the trigger relative to the unsuppressed samples.

ILAModule::GetData, ILAModule::ExportData - If sample suppression is used, the sample numbers specified are relative to the unsuppressed samples.

The following methods now return zero instead of an error if there is no acquisition data avaiable:

ILAModule::GetNumSamples

IDSOModule::GetNumSamples

Index